# **Salinas**–User's Notes

Garth Reese[*] Dan Segalman[†] Manoj K. Bhardwaj[‡]

Kenneth Alvin[§] Brian Driessen Kendall Pierson[¶] Timothy Walsh[‖]

Sandia National Laboratories
Albuquerque, NM 87185-0847

June 9, 2003

[*]Phone: 845-8640
[†]Phone: 844-0972
[‡]Phone: 844-3041
[§]Phone: 844-9329
[¶]Phone: 284-5894
[‖]Phone: 284-5374

Revision: 1.35

Date: 2003/06/03 21:48:44

Latest Formal Release: 1.1

## Abstract

Salinas provides a massively parallel implementation of structural dynamics finite element analysis, required for high fidelity, validated models used in modal, vibration, static and shock analysis of weapons systems. This document provides a users guide to the input for Salinas. Details of input specifications for the different solution types, output options, element types and parameters are included. The appendices contain detailed examples, and instructions for running the software on parallel platforms.

(this page intentionally blank)

**Salinas User Notes**

# Contents

(this page intentionally blank)

# Salinas

Salinas provides a massively parallel implementation of structural dynamics finite element analysis. This capability is required for high fidelity, validated models used in modal, vibration, static and shock analysis of weapons systems. General capabilities for modal, statics and transient dynamics are provided.

This document describes the input for the Salinas program. Examples of input specifications are scattered throughout the document. Appendix A provides several full input files. Appendix B provides instructions on invoking Salinas on a serial UNIX platform. Appendix C details how to execute Salinas on the ASCI-red machine, `janus`.

The name for Salinas is taken from a series of ancient Tewa Indian pueblos to the east of Albuquerque, New Mexico. These pueblos have been a source of culture and of salt for centuries. They were among the first settlements for Spanish explorers in the region.

## 1 Introduction – Input File

The input file contains all the directives necessary for operation of the program. These include information on the type of solution, the name of the exodus file containing the finite element data, details of the material and properties within the element blocks, which boundary conditions to apply, etc. Details of each of these sections are covered below.

Typically, the input file has an extension of "`.inp`", although any extension is permitted. If the "`.inp`" extension is used, **Salinas** may be invoked on the input without specifying the extension.

The input file is logically separated into sections. Each section begins with a keyword (**Solution**, **BLOCK**, etc), and ends with the reserved word **end**. Words within a section are separated with "white space" consisting of tabs, spaces, and linefeeds. Comments are permitted anywhere within the file, and follow the C++ convention, i.e. a comment begins with the two characters "//" and ends with the end of the line.[1]

Except for data within quotes, the input file is case insensitive. The software converts everything to lower case unless it is enclosed in quotes. Either the single quote `'` or the double quote `"` may be used. The quotes may be nested, e.g. `'a string with "embedded" quotes'`, but only with the other style mark.

[1] To be safe, define comments as "//" followed by a space.

The input parser supports nested includes. This is done using the **#include** command. This is the only command the parser recognizes. Files may be included to any depth. As an example,

```
#include english_materials
```

The **#include** may occur anywhere on the line (though for readability and consistency we recommend that it be the start of the line). The file name must immediately follow and should NOT be enclosed in quotes. Case sensitivity will be preserved. Summarizing, a minimum of two files are needed to run **Salinas** , namely, a text input file, e.g. *example.inp*, and an **Exodus** input file,[1] e.g. *example.exo*, which contains the finite element model. The finite element model is specified in *example.inp* as the geometry file (see section 2.9).

Each of the **Salinas** input sections is described in the following section.

# 2 The Salinas Input File

## 2.1 SOLUTION

The **solution** section determines which solution method, and options are to be applied to the model. The available solution types are shown in Table 1. Relevant options are shown in Table 3, and are described in section 2.2.

### 2.1.1 Multicase

All of the solution methods of table 1 may be a part of a multicase solution. This allows the user to specify multiple steps in a solution procedure. For example, there can be a static preload, a computation of the updated tangent stiffness matrix, and a linearized eigen analysis. The syntax for multicase solutions is similar to that for single cases, but each solution step is delineated by the "case" keyword. In addition, any of the modal solutions must be preceeded by an eigen analysis and eigen keywords are no longer recognized as part of the solution.

In a multicase solution, the system matrices (mass, stiffness and damping) will typically be computed only once. Matrix updates between solutions may be specified by selecting the **tangent** keyword (see section 2.1.18).

**Multicase Parameters.** Many of the solution parameters are specific to a particular solution type. For example, time step parameters are meaningless in a modal solution. However, some options apply more generally. These parameters, listed in Table 2, may be specified either above the case control sections, or within the section. The specification above the case control section is the default value. Specifications within the case sub-blocks apply only to that sub-block. In the example below, the `restart` options are thus "none" for most subcases, but "read" for the eigen analysis and auto for the linear transient.[2]

**Multicase Example.** In the example which follows, a nonlinear statics computation is followed by a tangent stiffness matrix update. The updated matrix is then used in an eigen analysis. Two sets of exodus output files will be written. Output from the statics calculation will be in files of the form *'example-nls.exo'*. Eigen results will be in the form *'example-eig.exo'*. The **tangent** solution normally produces no output in the exodus format.

[2]These features are not yet fully implemented in release 1.1. Currently only one restart or solver option is recognized for all solutions.

Table 1: Salinas Solution Types

| Solution Type | Description | Parameters |
|---|---|---|
| dump | form matrices only | |
| eigen | real eigensolution | nmodes, shift |
| eigenk | real eigensolution of K *(seldom useful)* | nmodes |
| subdomain_eigen | subdomain eigenanalysis *(ONLY for debug)* | nmodes |
| ceigen | unimplemented | |
| statics | static stress | |
| NLstatics | nonlinear statics | max_newton_iterations,tolerance num_newton_load_steps, update_tangent |
| checkout | skip large matrix and solves | |
| transient | implicit transient analysis | time_step, nsteps, nskip, rho, flush |
| NLtransient | implicit nonlinear transient analysis | time_step, nsteps, nskip, rho, flush, max_newton_iterations,tolerance |
| transhock | shock response spectra using direct implicit transient analysis | time_step, nsteps, nskip, flush srs_damp |
| modalfrf | frequency response using modal displacement or modal acceleration | nmodes, usemodalaccel, nrbms |
| directfrf | direct frequency response | |
| modaltransient | transient analysis using modal superposition | nmodes, time_step, nsteps, nskip, flush |
| modalranvib | random vibration using modal superposition | eigen parameters [noSVD] |
| modalshock | shock response spectra using modal approximate implicit transient analysis (unimplemented) | nmodes, time_step, nsteps, nskip, flush srs_damp |
| tangent | compute tangent matrices *(multicase only)* | |
| tsr_preload | thermal structural response *(multicase only)* | file |
| nox | NOX nonlinear solver library *(NLstatics and NLtransient problems only)* | NOX |

Table 2: Multicase Parameters
These parameters may be specified as defaults above the case specifications, or they may be specified for each subcase to which they apply.

| Parameter | Description | Options |
|---|---|---|
| restart | Restart options | see section 2.2.1 |
| solver | selection of solver | see section 2.2.2 |

```
Solution
   restart=none
   title='example multicase'
   case 'nls'
     nlstatics
     load=10
   case 'tangent'
     tangent
   case 'eig'
     eigen
     restart=read
   case 'trans'
     transient
       restart=auto
       time_step 1e-8 1e-6
       nsteps    100  4000
       flush 50
       rho=0.9
       load=20
END
```

The **case** keyword must always be followed by a label. The label is used in the output file name. The **case** keyword is also used to divide parameters of each solution type.

The **load** keyword is used within a solution step to indicate which loads to apply during a solution. In the example above, load '10' will be applied during the nonlinear statics calculation. During a multicase solution the **loads** section (found elsewhere in the file) will be ignored. See paragraph 2.11 for information on the **loads** section or paragraph 2.12 for information on the **load** section of the input file.

### 2.1.2 Eigen

The **eigen** keyword is needed to obtain the eigenvalues and mode shapes of a system. The parameters which can be specified for an eigensolution are shown in the table below. By default, if **nmodes** is not specified, a value of 10 is used.

| Parameter | Argument | Default |
|---|---|---|
| **nmodes** | *Integer* | 10 |
| **shift** | *Real* | 0 |

The **shift** parameter indicates the shift desired in an eigenanalysis. The shift value represents a shift in the eigenvalue space (i.e. $\omega^2$ space). The value to select is problem dependent, and only relevant for singular systems (i.e. floating structures). Please see the following discussion.

**Eigenanalysis of singular systems**

The eigenvalue problem is defined as,

$$(K - \omega^2 M)\phi = 0. \tag{1}$$

Where $K$ and $M$ are the stiffness and mass matrices respectively, and $\omega$ and $\phi$ are the eigen values and vectors to be determined. The problem may be solved using a variety of methods - the Lanczos algorithm is used in Salinas. In this method, a subspace is built by repeated solving equations of the form $Ku = b$. For floating structures, or structures with mechanisms, $K$ is singular and special approaches are required to solve the system. The two approaches used in Salinas are described below.

**Deflation.** If it is possible to identify the singularity in $K$, then the null vectors of $K$ are eigenvectors (with $\omega = 0$), and the system can be solved by insuring that no component of the null vectors ever occurs in $b$. This approach is equivalent to computing the pseudo inverse of $K$.

The strength of deflation is that if the eigenvectors can be determined exactly, the Lanczos algorithm is unaltered and the remaining vectors can be determined somewhat optimally. The difficulty is ensuring that we have correctly determined the eigenvectors, especially when mechanisms or multipoint constraints exist in the model. Determination of the eigenvectors is often a tolerance based approach that has not been as robust as we would like.

**Shifting.** The second method involves solution of a modified (or shifted) eigenvalue problem.

$$((K - \sigma) - \mu M)\,\phi = 0. \tag{2}$$

> This system has the properties that the eigenvectors, $\phi$, are unchanged from the original equation, and the eigenvalues, $\mu$, are simply related to the original values. Namely, $\mu = \omega^2 - \sigma$.
>
> The shifted problem benefits from the fact that $K - \sigma M$ can be made nonsingular (except in very rare situations). This is done by choosing $\sigma$ to be a large negative value. Unfortunately, the Lanczos routine convergence is affected if $\sigma$ is chosen to be too far from zero[3]. A reasonable value is $\sigma = -\omega_{elas}^2$, where $\omega_{elas}$ is the expected first nonzero (or elastic) eigenvalue.

On serial platforms we support only the shifted method. Because of the higher accuracy of direct solvers, a small negative shift is normally sufficient to solve the problem. This shift (usually -1) is computed automatically. We do not recommend that you override the defaults.

When using the FETI solver on parallel platforms both methods are available. If deflation is used, user input (and careful evaluation) may be required to ensure that all global rigid body modes have been properly identified. The relevant FETI parameters are `rbm` and `grbm_tol` as described in appendix D.

The shifted eigenvalue problem has proven to be more robust for many complex problems. Set the `grbm_tol` to a small value (e.g. 1e-20), and manually enter a negative shift. The output should still be examined to insure that no global rigid body modes are detected.

If the model is not floating and has no mechanisms, the system is not singular, and no shift should be used (as it may slow convergence).

**Example**

A **SOLUTION** section for an eigenanalysis with a **shift** of $-10^6$ , will look like the following, if 12 modes are needed. This shift would be appropriate for a system where the first elastic mode is approximately 150Hz.

```
Solution
  eigen
  nmodes 12
  shift -1.0e6
end
```

[3]If $\sigma$ is too large a negative value, many solves will be required to determine the eigenvalues (which consequently slows convergence). Another consequence is that often not all redundant, zero eigenvalues may be found. They may be found by reducing the shift, tightening tolerances or by restarting.

### 2.1.3 Eigenk

The **eigenk** keyword is used to obtain the eigenvalues and eigenvectors of the stiffness matrix of the model. This is equivalent to **eigen** if the mass matrix is equal to the identity matrix. The same parameters apply.
**IT IS CURRENTLY ONLY AVAILABLE ON SERIAL PLATFORMS.**

### 2.1.4 Subdomain_Eigen

The **subdomain_eigen** keyword is used to obtain the eigenvalues and eigenvectors of the mass and stiffness matrix of the model on a subdomain basis. This is useful mainly for debugging distributed solutions. It is obviously decomposition dependent, and has no physical meaning.

### 2.1.5 Ceigen

The **Ceigen** keyword is used to select complex eigen analysis. This computes the solution to the quadratic eigenvalue problem,

$$\left(K + D\lambda + M\lambda^2\right) u = 0 \tag{3}$$

Currently this is not implemented. We expect it in release 1.2.

### 2.1.6 Statics

The **statics** keyword is required if a static solution is needed, i.e. the solution to the system of equations $[K]\{u\} = \{f\}$. An example **SOLUTION** section is shown below.

```
Solution
  title 'Example of a statics solution'
  statics
end
```

### 2.1.7 NLStatics

The **NLstatics** keyword is required if a nonlinear static solution is needed, i.e. the solution to the system of equations $[K]\{u\} = \{f\}$, where $K$ is now a function of $u$. The following table gives the parameters needed for nonlinear static analysis.

| Parameter | Argument | Default |
|---|---|---|
| **max_newton_iterations** | *Integer* | 100 |
| **tolerance** | *Real* | 1e-6 |
| **num_newton_load_steps** | *Integer* | 1 |
| **update_tangent** | *Integer* | 101 |

Four parameters control the conventional Newton method. Newton methods are nonlinear solution algorithms employed to solve the residual force equations. The residual vector, $r$, is the difference between the internal force vector, it p and the external force vector, it f.

$$r = p - f = 0 \tag{4}$$

The internal force vector is a function of the structural displacements. External forces can also be a function of the structural displacements in the case of follower loads such as surface pressure loads.

The **tolerance** provides control over the completion of the newton iteration. Once the change in the L2 norm of *displacement* decreases below **tolerance**, the loop completes successfully. If the iteration count exceeds **max_newton_iterations**, the Newton loop is considered to have failed.

The **num_newton_load_steps** keyword controls the number of load steps used to incrementally step up to the final equilibrium position. Large loads may cause the Newton algorithm to diverge. If this occurs, increase the number of load steps applied. Displacements will be output after each load step which may be animated similar to transient dynamics simulations.

The **update_tangent** keyword controls how often the tangent stiffness matrix is rebuilt during the Newton iterations. The default is set to update the tangent stiffness matrix at the beginning of a load step only. Setting **update_tangent** to 1 is equivalent to using a full-Newton algorithm where the tangent stiffness matrix is rebuilt after each Newton iteration. For highly nonlinear (difficult) problems, this option may be optimal, but for most problems the extra cost incurred in recomputation and refactoring of the tangent stiffness matrix should be amortized over several solves. Note, for this option to improve Newtons method, the element types in the model have to have the tangent stiffness method implemented.

An example **SOLUTION** section is shown below.

```
Solution
  title 'Example of a nonlinear statics solution'
  nlstatics
  tolerance             = 1e-6
  max_newton_iterations = 100
  num_newton_load_steps = 10  // split external load into 10 increments
  update_tangent        = 1   // full-newton algorithm
end
```

### 2.1.8 Transient

The **transient** solution method is used to perform a direct implicit transient analysis. The following table gives the parameters needed for transient analysis.

| Parameter | Argument | Default | Purpose |
|---|---|---|---|
| **time_step** | *Real* | 1 | set the time step |
| **nsteps** | *Integer* | 100 | set the number of steps |
| **nskip** | *Integer* | 1 | set output frequency |
| **flush** | *Integer* | 50 | control file buffering |
| **rho** | *Real* | none - see below | select time integrator |

The parameters **time_step**, which defines the time integration step size, and **nsteps**, which defines the total number of integration steps, are required. The parameter **nskip** controls how many integration steps to take between outputting results and is optional. (It defaults to 1, which is equivalent to outputting all time steps).

The parameter **flush** controls how often the **Exodus** output file buffers should be flushed. Flushing the output insures that all the data that has written to the file buffers is also written to the disk. This parameter also controls the frequency of output of restart information if requested. Too frequent buffer flushes can affect performance. However, in a transient run, data integrity on the disk can only be assured if the buffers are flushed. A **flush** value of -1 will not flush the **Exodus** output file buffer until the run completes. The default value is to flush the buffers every 50 time steps.

We note that multiple time step values, along with the corresponding number of steps, can be specified for transient analysis. This can be useful for separating the simulation into a section of small time steps followed by a section of larger time steps, or vice versa. The following provides an example of the use of multiple time steps.

```
solution
  time_step     1e-5     1e-3
  nsteps        100      500
  nskip         10       1
end
```

In this case, the user requested 100 time steps of $\Delta t = 1E - 5$, followed by 500 time steps of $\Delta t = 1E-3$. There is no practical limit on the number of such regions that may be specified.

**Integrator selection**

Two time integrator schemes are available for direct time integration. The method and the parameters of the integrator are selected using the keyword **rho**. If this keyword is not found, the time integrator defaults to a standard Newmark-Beta integration scheme[4]. If the **rho** parameter is used, then the generalized alpha method[5] is used, and the value of the numerical damping is controlled by **rho**.

**** IMPORTANT ****

*Because of limited accuracy in the solvers, the Newmark-Beta integrator is conditionally unstable. If no damping is provided, it occasionally diverges as time progresses. This is described in a little more detail in the theory manual. Therefore it is strongly recommended that either proportional damping or numerical damping be used in all time integration.*

The parameter **rho** defines the Numerical damping of the generalized alpha method. **Rho** varies from 0 (maximal damping case) to 1 (minimal damping case). **If rho is not specified in the input file, the integrator defaults to the Newmark beta method.** Otherwise, the code uses the value of **rho** given by the user to compute the parameters needed for the generalized alpha method. Therefore, there is no value default for **rho**, as shown in the table above, since if it is not specified the code uses the Newmark beta method instead. If **rho** is specified to be greater than 1 or less than 0 an error message is printed. The three parameters **newmark_beta**, $\alpha_f$, and $\alpha_m$ in the generalized alpha method are computed automatically, given the value of **rho**, and thus these need not be specified by the user. More detailed information on the implementation, and references can be found in the description of the method in the Salinas program_notes and theory manual.

In order to achieve second order accuracy and unconditional stability, we must satisfy the following conditions.

$$\alpha_m < \alpha_f <= \frac{1}{2}$$

$$\gamma_n = \frac{1}{2} - \alpha_m + \alpha_f$$

[4]The Newmark-Beta integration is described in detail in most finite element text such as Cook or Hughes.

[5]Farhat, Crivelli, and Geradin, 'Implicit time integration of a class of constrained hybrid formulations - Part I: Spectral stability theory', CMAME, 125(1995), 71-107. Chung, J., Hulbert, G.M., 'A Time Integration Algorithm for Structural Dynamics with Improved Numerical Dissipation: The Generalized alpha method", J. Applied Mech., Vol.60, pp. 371-375.

$$\beta_n \geq \frac{1}{4} + \frac{1}{2}(\alpha_f - \alpha_m) \tag{5}$$

The code automatically computes these parameters such that they meet these criteria.

Unlike the proportional damping parameters, there is no direct relation between **rho** and an equivalent modal damping term. Numerical damping strongly affects only the highest frequency modes (which are non-physical anyway). We recommend a value of `rho=0.9` for most analyses.

### 2.1.9 NLTransient

The **NLtransient** solution method is used to perform a direct implicit nonlinear transient analysis. The following table gives the parameters needed for nonlinear transient analysis.

The nonlinear transient analysis is performed according to methods described in Hughes. A projector, corrector step is used. Note that for a linear system the NLtransient analysis will require two solves per time step.

| Parameter | Argument | Default |
|---|---|---|
| **time_step** | *Real* | - |
| **nsteps** | *Integer* | - |
| **nskip** | *Integer* | 1 |
| **flush** | *Integer* | 50 |
| **rho** | *Real* | Newmark beta |
| **max_newton_iterations** | *Integer* | 100 |
| **tolerance** | *Real* | 1e-6 |
| **num_newton_load_steps** | *Integer* | 1 |
| **update_tangent** | *Integer* | 101 |

The time step control parameters, **time_step**, **nsteps**, **nskip** and **flush** are described in the **transient** section above, section 2.1.8. The parameter **rho** is the same as described in the previous section. We note that, as in the case of linear transient analysis, multiple time steps can be specified in nonlinear transient analysis. The syntax for this is the same as described in the section on linear transient analysis.

Four parameters control the conventional Newton method used to solve the residual force equations. The **tolerance** provides control over the completion of the newton iteration. Once the change in the L2 norm of *acceleration* decreases below **tolerance**, the loop completes successfully. If the iteration count exceeds **max_newton_iterations**, the Newton loop is considered to have failed.

The **num_newton_load_steps** controls the number of load steps used to incrementally step up to the final equilibrium position. Large loads may cause the Newton algorithm to diverge. For nonlinear statics, it is recommended to increase the number of load steps. For nonlinear transient problems, if Newtons method diverges, either the tangent stiffness matrix has to be updated more often (see **update_tangent**) or the time-step should be decreased. The default value is 1 for both nonlinear statics and nonlinear transient solution methods.

The **update_tangent** controls how often the dynamic tangent stiffness matrix is rebuilt during the Newton iterations. The default is set to update the dynamic tangent stiffness matrix at the beginning of a load step. Setting **update_tangent** to 1 is equivalent to using a full-Newton algorithm where the dynamic tangent stiffness matrix is rebuilt after each Newton iteration. For highly nonlinear problems, some control of this option is recommended. Note, for this option to improve Newtons method, the element types in the model have to have the dynamic tangent stiffness method implemented.

### 2.1.10 Transhock

The **transhock** solution method is used to perform a direct implicit transient analysis followed by computation of the shock response spectra for the degrees of freedom in a specified node set (all node sets are defined in the Exodus file). The following table gives the parameters needed for transient shock analysis.

| Parameter | Argument |
|---|---|
| **time_step** | *Real* |
| **nsteps** | *Integer* |
| **nskip** | *Integer* |
| **srs_damp** | *Real* |

The parameters **time_step**, which defines the time integration step size, and **nsteps**, which defines the total number of integration steps, are required. The parameter **nskip** controls how many integration steps to take between outputting results and is optional. (It defaults to 1, which is equivalent to outputting all time steps).

The parameters **freq_step**, **freq_min**, and **freq_max** are used to define the frequencies for computing the shock response spectra. They are identified in the **frequency** section along with an application region (see section 2.8). The range of the computed frequency spectra is controlled by **freq_min** and **freq_max**, while

**freq_step** controls the resolution. The accuracy of the computed spectra is not dependent on the magnitude of **freq_step**. This parameter only controls the quantity of output.

The keyword **srs_damp** is a damping constant used for the shock response spectra calculation and is optional. It represents the damping for each single degree of freedom oscillator in the shock spectra computation. Its default value is 0.03.

Note: Currently, the shock spectrum procedure will only compute acceleration results. The options specified in the OUTPUT and ECHO blocks are used in the transient portion of the analysis, but are ignored for the post-processing of the transient results into shock spectra. Thus, if displacement, velocity, and/or acceleration is selected in the OUTPUT and/or ECHO sections for a shock spectra analysis, the results echoed to the output listing or the Exodus output file will be time history results as requested, but the only shock spectra results will be for acceleration response for the nodes in the specified node set. Furthermore, *the calculated shock spectra will only be echoed to the output listing; they are not output to the Exodus results file.* The shock spectra output options will be revised and improved in future releases.

### 2.1.11 Modalfrf

Option **modalfrf** is used to perform a modal superposition-based frequency response analysis. In other words, the modalfrf provides an approximate solution to the Fourier transform of the equations of motion, i.e.

$$\left(K + i\omega C - \omega^2 M\right) \overline{u} = \overline{f}(\omega)$$

where $\overline{u}$ is the Fourier transform of the displacement, $u$, and $\overline{f}$ is the Fourier transform of the applied force.

Two options are available for the modalfrf solution: the modal displacement method, and the modal acceleration method. In both cases the approximate solution is found by linear modal superposition. Once the modes have been computed, there is little cost in computation of the frequency response. The solution does suffer from modal truncation of course, but in the case of the modal acceleration method a static correction term partially accounts for the truncated high frequency terms. Thus, in general that method is more accurate than the modal displacement method. The most accurate, but also the most computationally expensive approach is the **directfrf** method described in section 2.1.12.

For the modal displacement method, the relation used for modal frequency response is given below.

$$\overline{u}_k(\omega) = \sum_j \frac{\phi_{jk}\phi_{jm}\overline{f}_m(\omega)}{\omega_j^2 - \omega^2 + 2i\gamma_j\omega_j\omega}$$

Here $\overline{u}_k$ is the Fourier component of displacement at degree of freedom $k$, $\phi_{jk}$ is the eigenvector of mode $i$ at dof $k$, and $\omega_j$ and $\gamma_j$ represent the eigenfrequency and associated fractional modal damping respectively.

For the modal acceleration method, the procedure for computing the modal frequency response is more complicated. The response is split into the rigid body contributions, and the flexible contributions. The number of global rigid body modes must be specified in the input file. For details on the theory, we refer to the theory manual.

The force function must be explicitly specified in the load section, and MUST have a “function” definition. Note that the force input provides the real part of the force at a given frequency, i.e. it is a function of frequency, not of time. At this time, we do not provide a way to input the imaginary component of the force.

The following table gives the parameters needed for **modalfrf** section.

| Parameter | Argument |
|---|---|
| **nmodes** | *Integer* |
| **usemodalaccel** | - |
| **nrbms** | *Integer* |

The **nmodes** parameter controls the eigenanalysis (see section 2.1.2). The optional keyword, **usemodalaccel**, is used to determine whether to use the modal displacement or the modal acceleration method. If this keyword is specified, modal acceleration is used, otherwise the modal displacement method is invoked. If **usemodalaccel** is used, then the number of global rigid body modes must be specified using **nrbms**.

The parameters **freq_step**, **freq_min**, and **freq_max** are used to define the frequencies for computing the shock response spectra. They are identified in the **frequency** section along with an application region (see section 2.8). The range of the computed frequency spectra is controlled by **freq_min** and **freq_max**, while **freq_step** controls the resolution. The accuracy of the computed spectra is not dependent on the magnitude of **freq_step**. This parameter only controls the quantity of output.

### 2.1.12 Directfrf

Option **directfrf** is used to perform a direct frequency response analysis. In other words, we compute a solution to the Fourier transform of the equations of motion, i.e.

$$\left(K + i\omega C - \omega^2 M\right) \overline{u} = \overline{f}(\omega)$$

where $\overline{u}$ is the Fourier transform of the displacement, $u$, and $\overline{f}$ is the Fourier transform of the applied force. The method used is to compute the frequency dependent matrix $A(\omega) = K + i\omega C - \omega^2 M$, and frequency component of the force at each frequency point at the output. The matrix equation is then solved once per frequency point. When a direct solver is used, this means that a complex factorization must be performed once per output. This can be very time consuming, and the **modalfrf** may be a better option for many situations (see section 2.1.11).

The force function must be explicitly specified in the load section, and MUST have a "function" definition. Note that the force input provides the real part of the force at a given frequency, i.e. it is a function of frequency, not of time. At this time, we do not provide a way to input a complex force.

The parameters **freq_step**, **freq_min**, and **freq_max** are used to define the frequencies for computing the directfrf. They are identified in the **frequency** section along with an application region (see section 2.8). The range of the computed frequency response is controlled by **freq_min** and **freq_max**, while **freq_step** controls the resolution.

Note that, currently, directfrf is only a serial implementation. The parallel implementation is scheduled for implementation in release 1.2.

### 2.1.13 Modaltransient

Option **modaltransient** is used to perform a modal superposition-based implicit transient analysis. The following table gives the parameters needed for **modaltransient**. Damping for the model is defined in section 2.22.

| Parameter | Argument | default |
|---|---|---|
| **time_step** | *Real* | none |
| **nsteps** | *Integer* | none |
| **nskip** | *Integer* | 1 |
| **load** | *Integer* | *sec 2.12* |

The parameters **time_step**, which defines the time integration step size, and **nsteps**, which defines the total number of integration steps, are required. The optional parameter **nskip** controls how many integration steps to take between outputting results. (It defaults to 1, which is equivalent to outputting all time steps). Time dependent loadings are applied by referencing the appropriate **load** and **function** sections (see 2.12 and 2.19).

Modal transient should normally be executed as a later step of a multicase solution, where previous steps computed the eigenvalue response. However, for compatibility with earlier formats, **modaltransient** can be called as a single step solution (see section 2.1.2). In that case the following eigen value parameters are also required. Note that in a single step solution (with no case structure), no **load** keyword is required, but a **loads** section must exist in the file (see section 2.11).

| Parameter | Argument |
|---|---|
| **nmodes** | *Integer* |
| **shift** | *Real* |

### 2.1.14 Modalshock

The **modalshock** solution method is used to perform a modal superposition-based implicit transient analysis followed by computation of the shock response spectra for the degrees of freedom in a specified node set. The following table gives the parameters needed for **modalshock**.

| Parameter | Argument |
|---|---|
| **nmodes** | *Integer* |
| **time_step** | *Real* |
| **nsteps** | *Integer* |
| **nskip** | *Integer* |
| **srs_damp** | *Real* |

The **nmodes** parameter controls the modal solution described in section 2.1.2. The time stepping parameters **time_step**, **nsteps** and **nskip** are described in the transient section (2.1.8).

The parameters **freq_step**, **freq_min**, and **freq_max** are used to define the frequencies for computing the shock response spectra. They are identified in the **frequency** section along with an application region (see section 2.8). The range of the computed frequency spectra is controlled by **freq_min** and **freq_max**, while **freq_step** controls the resolution. The accuracy of the computed spectra is not dependent on the magnitude of **freq_step**. This parameter only controls the quantity of output.

The optional parameter **srs_damp** is a damping constant used for the shock response spectra calculation. Its default value is 0.03. Damping for the model is defined in section 2.22.

### 2.1.15 Modalranvib

Option **modalranvib** is used to perform a modal superposition-based random vibration analysis in the frequency domain. The solution computes the root mean square (RMS) outputs (including the von mises stress) for a given input random force function. The resulting power spectral density functions may also be output for locations specified in the "frequency" section. The forcing functions (one for each input) must be explicitly specified in the load section, and MUST have a "matrix-function" definition (see section 2.20).

The following table gives the **solution** parameters needed for **modalranvib** analysis.

| Parameter | Argument |
|---|---|
| **nmodes** | *Integer* |
| [**noSVD**] | N/A |
| **lfcutoff** | *Real* |
| **keepmodes** | *Integer* |

The **nmodes** parameter controls the eigenanalysis (see section 2.1.2). All keywords associated with eigen analysis are appropriate and available. It is recommended that the eigenanalysis be performed as the first step of a multicase solution.

The *optional* keyword **noSVD** determines the method used to compute the RMS von Mises stress output. If **noSVD** is specified, then the simpler method which does not use a singular value decomposition is used. However, this method provides no information about the statistics of the stress. Only the RMS value is reported.

The *optional* keyword **lfcutoff** provides a low frequency cutoff for random vibration processing. Usually, rigid body modes are *not* included in this type of calculation. The **lfcutoff** provides a frequency below which the modes are ignored. The default for this value is 0.1 Hz. Thus, by default rigid body modes are not included in random vibration analysis. A large negative value will include all the modes.

The *optional* keyword **keepmodes** is a method of truncating modes. By default, its value is **nmodes**. If a value is provided, the modes with the lowest modal activity will be truncated until only **keepmodes** remain. Note that this is a much different truncation procedure than simply truncating the higher frequency modes. Modal truncation is important because all of the operations compute responses that require order $N^2$ operations. Even if **keepmodes** is not entered, modes with modal activity less than 1 millionth of the highest active mode will be truncated.

The parameters **freq_step**, **freq_min**, and **freq_max** are used to define the frequencies for computing the random vibration spectra. They are identified in the **frequency** section along with an optional application region (see section 2.8). The range of the computed frequency spectra is controlled by **freq_min** and **freq_max**, while **freq_step** controls the resolution. The accuracy of the computed spectra *does* depend on the magnitude of **freq_step** since it is used in the frequency domain integration.

In random vibration, the **frequency** block serves two purposes. First, it is used for the integration information for the entire model. Thus $\Gamma_{qq}$ for the referenced papers[6] is integrated over frequency and used for all output. In addition, if an output region is specified in the **frequency** block, output acceleration and displacement power spectra may be computed for the given region at the required frequency points. At this time, only "acceleration" and/or "displacement" may be specified in the frequency block for random vibration analysis. This output is described in more detail below.

Random vibration analysis is a little trickier than most input. A number of blocks must be specified.

1. The **solution** block must have the required input for eigen analysis, and the keyword **modalrand**.

2. The **ranloads** block contains a definition of the spectral loading input matrix and the loadings. Note that the input, $S_{FF}$ is separated into frequency and spatial components. The spatial component is specified here using **load** keywords. See section 2.13. The spectral component is referred to here, but details are provided in the matrix-function section.

3. The **matrix-function** section contains the spectral information on the loading. It references functions for the details of the load. The real and imaginary function identifiers for this input are specified here (2.20).

4. There must be a **function** definition for each referenced spectral function. Functions of time or frequency are further described in section 2.19.

5. There must be a **frequency** block that is used for integration and optionally also for output of displacement and acceleration output. See section 2.8.

[6]Reese, Field and Segalman, *A Tutorial on Design Analysis Using von Mises Stress in Random Vibration Environments* Shock and Vibr. Digest, Vol. 32, No. 6, Nov 2000.

6. As an undamped system is singular, some type of **damping** block information needs to be provided. Modal damping terms are required. See section 2.22.

7. **Boundary** conditions are supplied in the usual way, but the standard **LOADS** block is replaced by the input in the ranloads section. The loads block will be quietly ignored in random vibration analysis.

8. The **output** and **echo** sections will require the keyword **vrms** for output of RMS quantities.

All other input should remain unchanged.

**Power Spectral Densities.** One output from the random vibration analysis is a power spectral density or PSD (for displacement or acceleration). The power spectral density is a measure of the output content over a frequency band, and usually measured in units of $cm^2/Hz$ or some similar unit. Acceleration PSDs are often measured in units of $g^2/Hz$.

Like the input cross spectral forces, the output quantities are hermitian, with 9 independent quantities at each frequency, at each output node for each type of output. Details of how these quantities are transformed in alternate coordinate systems are outlined in the theory manual. The matrix quantities are diagramed below. Quantities are output in the order $A_{xx}$, $A_{yy}$, $A_{zz}$, $A_{zx}$, $A_{zy}$, $A_{xy}$, $A_{zxi}$, $A_{zyi}$, $A_{xyi}$.

$$\begin{bmatrix} A_{xx} & A_{xy} + iA_{xyi} & A_{xz} + iA_{xzi} \\ A_{xy} - iA_{xyi} & A_{yy} & A_{yz} + iA_{yzi} \\ A_{xz} - iA_{xzi} & A_{yz} - iA_{yzi} & A_{zz} \end{bmatrix}$$

Because the inputs are specified in terms of force cross-correlation functions, the standard procedure for applying loads often involves application of a large concentrated mass at the input location. The force may then be applied to the mass and the acceleration determined from $a = f/m$, where we assume that $m$ is much larger than the mass of the remainder of the structure. Some confusion can arise in the scaling of the force.

The output PSD for acceleration is defined as follows.

$$G_{ij} = H^{\dagger}_{ki} S_{kl} H_{lj} \tag{6}$$

$$< a_i a_j > = H^{\dagger}_{ki} < f_k f_l > H_{lj} \tag{7}$$

where $H_{lj}$ is the transfer function giving $a_j/f_l$.
Consider a single input, i.e. $k = l$, and with $f_k = m_k a_k$.

$$\begin{aligned} G_{ij} &= H^{\dagger}_{ki} < m_k a_k a_k m_k > H_{lj} && (8) \\ &= (m_k^2) H_{ki} < a_k a_k > H_{kj} && (9) \end{aligned}$$

Thus, the acceleration PSD must be multiplied by the square of the mass to get the force PSD. Note that **Salinas** uses the scale factor in the spatial force distribution, so the scale factor in **Salinas** should be $m_k$.

### 2.1.16 Checkout

The **checkout** solution method tests out various parts of the code without forming the system matrices or solving the system of equations. This solution method may be used to check input files for consistency and completeness on a serial platform before allocating expensive resources for a full solution.

### 2.1.17 Dump

The keyword **dump** will cause **Salinas** to form matrices only and no solution will be obtained.

### 2.1.18 Tangent

The **tangent** solution step is only relevant as part of a multicase solution (see paragraph 2.1.1). It forces an update of the tangent stiffness matrix. It is typically used following a nonlinear solution step to insure that the following step begins using the tangent stiffness matrices computed from the previous result. However, it may also be used following a linear solution step, in which case the stiffness matrix is recomputed based on the current value of displacement.

The tangent stiffness matrix is assembled at the subdomain level from computations at the element level. It represents the partial derivative of the force with respect to the displacement, i.e.

$$K_{tangent} = \frac{\partial f}{\partial u} \qquad (10)$$

In eigen analysis, the tangent stiffness matrix replaces the linear stiffness matrix in the eigenvalue equation. This permits computation of modal response following a preload. In nonlinear transient dynamics, the tangent stiffness matrix is used in the Newton (or other) iteration scheme used to reduce force residuals.

### 2.1.19 TSR_Preload

The **tsr_preload** solution method reads an exodus file with a previously computed Thermal Structural Response (TSR) into Salinas for a subsequent statics or transient dynamics analysis. This is not a fully coupled calculation. Rather, stress results are read from the file, an equivalent internal force is computed, and that internal force is combined with the applied force throughout the transient run. **Tsr_preload** may only be specified as part of a multicase solution, and it must be followed by a transient dynamics solution (see paragraphs 2.1.1 and 2.1.8 respectively).

Note that since the stresses are actually converted into a force, and since there is no immediate deformation in transient dynamics, the elastic stresses output by Salinas will be very small initially, i.e. they will not contain a contribution from the thermal stress. However, at large times, the deformation from the internal force will result in an elastic stress opposite to that of the thermal stress. There is currently no method of recovering the input thermal stress as an output quantity.

The **tsr_preload** solution method is considered to be a temporary solution to a more complicated problem. In the future, TSR analysis will involve coupling to other mechanics codes.

The following table gives the **solution** optional parameter used in **tsr_preload** analysis.

| Parameter | Type | Argument |
|---|---|---|
| **file** | *string* | exodus_file_name |

The `exodus_file_name` is a string that points to the file containing the stress results from the TSR calculation. If no file keyword is provided, the data is expected in the input genesis file, i.e. the **geometry_file** specified in the **FILE** section (see paragraph 2.9). Currently, for parallel execution, the data must be specified in the genesis file, as the file name is not properly parsed for spread files.

Data in the exodus file must strictly match these criteria. There must be only one time step in the result. That time step must have a number of different element fields defined. These correspond to the six stresses and up to 27 different integration points of a hex20. Other solid elements are also supported. For those elements only the number of integration points applicable to that element are used. Unused integration values will be ignored. If in doubt, provide the extra integration data as missing integration points do NOT provide an error - rather they set the value to zero. Shell and beam type elements are not supported in tsr_preload.

The labels for the stresses must be as shown in the table below. In each case, replace %d with an integer representing the integration point value (0 to 26).

| Name | Definition |
|---|---|
| SIGXX_%d | $\sigma_{xx}$, the $xx$ component of stress |
| SIGYY_%d | $\sigma_{yy}$, the $yy$ component of stress |
| SIGZZ_%d | $\sigma_{zz}$, the $zz$ component of stress |
| SIGYZ_%d | $\sigma_{yz}$, the $yz$ component of stress |
| SIGXZ_%d | $\sigma_{xz}$, the $xz$ component of stress |
| SIGXY_%d | $\sigma_{xy}$, the $xy$ component of stress |

The following is an example solution section for a TSR preload followed by transient dynamics.

```
SOLUTION
   title 'Pure bending from initial stress'
   case tsr
        tsr_preload
        load 1
   case 'trn'
      transient
      time_step 1.e-6
        nsteps 3
        nskip 1
      load 2
END
```

If executed on a file with `geometry_file='example.exo'`, this will produce two output files, `example-tsr.exo` and `example-trn.exo`. The first of these has very little useful information. The second will contain the displacements (or other variables) from the transient analysis.

### 2.1.20 NOX

The **nox** solution option allows the nonlinear solution procedure for either NLstatics or NLtransient problems to be driven by the NOX nonlinear solver library. Currently, Salinas can be built to run with NOX on only selected development platforms. These include serial execution on PC platforms running under Linux and parallel execution on solaris machines. The builds are achieved using the makefiles make.linux.nox and make.mpsolaris.7.nox, respectively. For such builds, the NOX solver is not used by default but can be turned on by including the **nox** keyword as follows.

An example **SOLUTION** section is shown below.

```
Solution
  title 'Example of a nonlinear statics solution'
  nlstatics
  tolerance            = 1e-6
  max_newton_iterations = 100
  num_newton_load_steps = 10  // split external load into 10 increments
  update_tangent        = 1   // full-newton algorithm
  nox  // use NOX nonliner solver
end
```

This option has no effect other than to invoke additional parsing of the input deck for problems other than NLstatics and NLtransient. Simply using NOX in this way causes the default solution procedure to be used. This corresponds to the Newton-based approach that is used in Salinas as described in sections 2.1.7 and 2.1.9). Other solution strategies provided by the NOX library are described in section 2.23.

It should be noted that the NOX solver interface to Salinas is new and will likely contain some bugs. If found, please e-mail a description of the bug to Russell Hooper at rhoope@sandia.gov. At this time, there is a known bug with parallel execution for problems involving element types of dimension two or three. This is currently being addressed. In addition, support for other platforms is also anticipated to be provided shortly.

## 2.2 Solution Options

The options described in Table 3 and in the following paragraphs are part of the **Solution** section in the input file. None of the keywords are required. Note that in multicase solutions most of these parameters may be applied separately within the subcase (see section 2.1.1).

Table 3: Salinas Solution Options

| Option | Description | Parameters |
|---|---|---|
| restart | restart options | **none**, read, write or auto |
| lumped | Use lumped mass matrices | none |
| solver | Identify solver used | “auto” |
| constraintmethod | method of applying MPCs | Lagrange or Transform |

### 2.2.1 ReStart – option

Option **restart** controls restart file processing. Restart files permit the solution to be saved for later use. Only a limited capability is provided, but it is intended to meet most of the typical needs for structural dynamics. Note that the restart files are independent of the exodus output, but the restart options may significantly affect the exodus outputs. Application of restarts in specific sections is detailed in the following paragraphs.

There are four values associated with this option.

**none** indicates that restart files will be ignored. They will be neither read, nor written. Existing restart files will not be altered in any way. Restart=none is the default selection if no restart options are entered in the solution block.

**read** indicates that existing restart files will be read, but no output restart files will be written. If the restart files do not exist, a fatal error will result.

**write** indicates that existing restart files will be ignored, but restart files will be written.

**auto** is a combination of read and write. However, unlike read, the existence of previous restart files is optional, i.e. there will be no error message if there are no existing restart files. Invalid restart files will produce a warning, but not a fatal error.

Restarts are designed to insure accuracy of the solution. However, restarts in Salinas are not transparent in the sense that there will be small differences in two solutions to a problem when one solution involves a restart. Restarts may also have an expense. For example, the FETI solver uses an acceleration technique where the values of previous solutions are used as a starting place for new solves. The information associated with previous solutions is not stored in the file.

For transient dynamics, the state of the machine at the most recent time step is recorded. To avoid problems with corruption of a database, the three vectors (disp, velocity, acceleration) are recorded at each time step, but on alternate locations in the file. If previous exodus files exist, they will be appended. Data is written at the same interval as the exodus output.

### 2.2.2 Solver

As Salinas evolves, various solvers are available for computation of the solution. Each solver brings with it different capabilities and sometimes unwanted features. Currently available solvers are listed in the following.

**AUTO** Use the best known solver. Generally this is recommended. The matrix of solvers versus solution types is messy, and generally the best solution will be found by using this option. For example, there is no need to change the solver as you move from serial to parallel solutions.

**CLASP** Under development by Clark Dohrman, this is a multigrid solver. It is not currently considered a production solver.

**FETI-DP** This solver is the workhorse for parallel solutions. A full description of the solver is beyond the scope of this users manual (references are on the web). FETI-DP was developed by Charbel Farhat and Kendall Pierson. It is very scalable, and robust. Multipoint constraints are handled using Lagrange multipliers. The parallel solution process must be used with the solver, but it can be reduced to a single subdomain. Care must be used to insure that subdomains are mechanism free.

**FETI-DPC** An evolution of FETI-DP, this solver adds the capability to compute constraints within the solver. NOT CURRENTLY AVAILABLE.

**FETI-H** This is another variant of FETI, designed specifically for complex solutions. It is used only in the solution of direct frequency response functions.NOT CURRENTLY AVAILABLE.

**Genfac** This is the only solver currently available in serial solutions. It is a direct solver, and is part of sparspak developed by Esmond Ng. The solver is fairly robust, but may fail for singular systems. It occasionally has problems for very small systems. Originally written as a Cholesky decomposition, it has been extended to compute $LDL^T$. Constraints are eliminated using a transformation matrix method.

**Prometheus** Developed by Mark Adams, this is another multigrid solver. It has some very nice features such as augmented Lagrange constraint handling. NOT CURRENTLY AVAILABLE.

**SuperLU** This package, available from NERSC, provides both serial real and complex solutions. In salinas, the complex version is used for solution of direct FRFs.

**NOX** This package, curently being developed at Sandia and available at http://software.sandia.gov, can be used in conjunction with any of the available linear solvers to drive the nonlinear solution procedure for NLstatics and NLtransient problems. It is currently available on only a small subset of platforms.

Generally no user input is required for specification of a solver. Indeed, up to version 1.0.5 of Salinas, only one solver was ever available at any time (i.e. we built separate executables if another solver was desired). Usually the specification can be left off, or specified as “auto”. If a solver is requested and unavailable, a warning will be issued, and “auto” will be selected.

The solver may be specified as a default (above the `case` keywords as detailed in section 2.1.1), or it may be individually specified within the case framework. The default value is “auto”. In the example shown below FETI-DP will be used for the eigen analysis, FETI-DPC for transient dynamics, and the “auto” selection for the direct frequency response. If “input” is specified in the “echo” section (see section 2.5) then the solver information will be echoed to the results file.

```
SOLUTION
  solver=auto
  case eig
    eigen nmodes=50
    solver=feti-dp
  case nlt
    nltransient
    solver=feti-dpc
    (other parameters)
  case frf
    directfrf
 END
```

### 2.2.3 Lumped – option

Option **lumped** in the **SOLUTION** section causes **Salinas** to use a lumped mass matrix, and not consistent mass matrix, in the analysis.

### 2.2.4 Constraintmethod – option

The **constraintmethod** option is defined in the **SOLUTION** section to indicate how multipoint constraints (MPC) will be applied. The selections for applying MPCs are are **Lagrange** and **Transform**. These methods are explained in detail on pp. 272-278 in Ref. 2.

The **constraintmethod** is currently superfluous. When using the `FETI` solver, a Lagrange multiplier method is the only method available. When using the serial solvers, the only available method is **Transform**.

## 2.3 PARAMETERS

This *optional* section provides a way to input parameters that are independent of the solution method or solver. Only one **parameter** section is recognized in each file. The parameters and their meanings are listed below.

**WtMass** This variable multiplies all mass and density on the input, and divides out the results on the output. It is provided primarily for the english system of units where the natural units of mass are actually units of force. For example, the density of steel is $0.283 lbs/in^2$, but "lbs" includes the units of g, 386.4 $in/s^2$. Using a value of wtmass of 0.00259 (1/386.4), density can be entered as 0.283, the outputs will be in pounds, but the calculations will be performed using the correct mass units.

Salinas, like most finite element codes, does not manage the *units* of the analysis. The selection of a consistent set of units is left to the analyst. For example, if the analyst uses the SI system (Kg,m,s) the units of pressure must be Pascals. Frequencies are reported in Hz. For micromachines these units are quite awkward. It may be better to use units of grams, millimeters and microseconds. The analyst must insure that all material properties and loads are converted to these units.

Some examples of useful units are shown in Table 4.

Table 4: Some useful combinations of units.

| length | mass | time | wtmass | density | force | modulus |
|---|---|---|---|---|---|---|
| $m$ | Kg | sec | 1 | Kg/$m^3$ | $N$ | N/$m^2$ or Pa |
| ft | slug | sec | 1 | slug/$ft^3$ | $lbf$ | $lb/ft^2$ |
| ft | $lbm$ | sec | 1/32.2 | $lbm/ft^3$ | $lbf$ | $lb/ft^2$ |
| in | $lbm$ | sec | 1/386.4 | $lbm/in^3$ | $lbf$ | psi |
| $mm$ | $\mu g$ | $\mu s$ | 1 | Kg/$m^3$ | $N$ | MN/$m^2$ or MPa |
| $mm$ | $g$ | sec | 1/1000 | g/$cm^3$ | $\mu N$ | N/$m^2$ or Pa |

**NegEigen** Unconstrained structures have zero energy modes which may evaluate to small negative numbers due to machine round off. The eigenvalues and associated eigenfrequencies are reported as negative numbers in the results files. However, many post processing tools (such as *ensight*) require non-negative frequencies. By default, Salinas converts all negative eigenvalues to near zero values in the output exodus files[7]. To retain the negative eigenvalues in the output file, select parameter **NegEigen**.

**OldBeam** This option is provided for backwards compatibility with older beam models. Early Patran models using the exodus preference numbered the attributes incorrectly. The first versions of Salinas used that numbering. With the new numbering the code had to change. Providing "oldbeam" in the parameters section selects the old numbering. The new numbering will be used by default. At some point in the future, we plan to eliminate this option.

Table 5: Beam Attribute Ordering

| Atttribute | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| old numbering | area | orientation | | | I1 | I2 | J |
| new numbering | area | I1 | I2 | J | orientation | | |

**eig_tol** This is the tolerance used by ARPACK for eigensolution. If not provided, an automatic value is used.

**MaxResidual** This is a tolerance used to check the rigid body mode vectors calculated by `FETI`. If this residual on the rigid body mode vector is larger than this tolerance, **Salinas** will abort. The default value is 1.0.

**LinkStiffness** This option makes it easier for some solvers to properly compute the response when there are many rigid links. At present, only RBARS and RRODS (see sections 3.27 and 3.26) are affected. The option causes **Salinas** to compute additional stiffness terms that would be associated with a beam (or truss) in place of the rigid element. Since the constraint limits the deformation to zero, there is no affect on the final solution, but the solution process can be significantly simplified since singularities are removed from the stiffness matrix. Specify `LinkStiffness=yes` or `LinkStiffness=no`. The default value is `yes`, which means the additional stiffness terms are used.

**SaveElemMatrices** In nonlinear transient dynamics, the element forces are recomputed at each iteration. For nonlinear elements, this is appropriate. However, for linear elements the calculation involves recomputation of the same

[7]Because many postprocessing tools are written for transient dynamics, they expect monotonically increasing, positive values for the time. Since eigenvalues are written in the time columns of the output file, they are converted to be monotonically increasing, positive values. Note that the numerically computed eigen frequencies are also stored as global variables in the file

linear element matrices. The keyword **SaveElemMatrices** signals the application to save the element stiffness, mass and damping matrices so these force calculations can be streamlined. The speedup can be substantial, but some additional memory is required. Elements that use viscoelastic materials always save element matrices.

**TangentMethod** The tangent stiffness matrix may be used in a full Newton update in nonlinear statics or transient dynamics (see sections 2.1.7 and 2.1.9). By default, each of the elements can compute it's own tangent stiffness matrix. There are cases (particularly when elements are under development) when it is better to use a tangent matrix computed from finite difference methods. There are three possible values for this keyword.

**TangentMethod=element** The standard element method.

**TangentMethod=difference** Use finite difference.

**TangentMethod=compare** Use the standard method, but also compute the matrix by the difference method. Unless "none" is specified in the ECHO section (2.5), output of the difference of every element matrix in the model will be sent to the results file.[8]

Example,

```
Parameters
  saveelemmatrices
  WtMass=0.001
End
```

## 2.4 FETI

This *optional* section provides a way to input parameters specific to the Finite Element Tearing and Interconnecting[3] (FETI) solver, if used. If the FETI solver is not used, this section is ignored. It includes the following parameters, shown in Table 6, and options. For those options which are strings, only enough of the string to identify the value is required. The defaults are shown in the following example.

```
FETI
      rbm  geometric
      scaling no
```

[8]In parallel solutions the results file is written only for the first processor unless the "subdomains" option is specified in the echo section (2.5).

Table 6: FETI Section Options

| Variable | Values | Description |
|---|---|---|
| rbm | Algebraic/Geometric | rigid body mode method |
| scaling | Yes/No | scaling method |
| preconditioner | LUMped/DIRichlet | (both may be used) |
| max_iter | *Integer* | maximum number iterations |
| solver_tol | *Real* | |
| stag_tol | *Real* | Used to detect stagnation |
| orthog | *Integer* | max number of orthog. vectors |
| rbm_tol_svd | *Real* | SVD tolerance in rigid body modes |
| rbm_tol_mech | *Real* | mechanical tolerance in rbm |
| projector | **Standard**/Q | projector |
| level | 1 | feti1 (feti2 not implemented) |
| corner_dimensionality | *Integer* | 3 or **6** dofs/corner |
| corner_algorithm | *Integer* | 1, 3, 5-8 |
| corner_augmentation | *String* | **"none"**, "subdomain", "edge" |
| local_solver | AUto, SKyline, SParse, serial_sparse | solver for local LU decomp |
| precondition_solver | Same as local_solver | solver for preconditioner Only used if using dirichlet preconditioner |
| coarse_solver | AUto, SKyline, SParse PSparse, serial_skyline, serial_sparse | solver for coarse $G^TG$ problem (psparse is parallel sparse) |
| grbm_tol | *Real* | tolerance for rigid body detection in $G^TG$ |
| prt_summary | Yes/**No** | print summary timer information |
| prt_rbm | Yes/**No** | print # rbm in each subdomain |
| prt_debug | integer | debug output. values **0=none**, 1-3 |
| bailout | | if set, the solver will continue even if the solution is not converged at each intermediate solve |
| mpc_method | *Integer* | **0**=Lagrange multipliers everywhere 1=Local elimination where possible |

```
        preconditioner dirichlet
        max_iter 400
        solver_tol 1.0e-5
        orthog 1000
        rbm_tol_svd 1.0e-10
        rbm_tol_mech 1.0e-8
        projector standard          // ignored in dp
        level 1                     // ignored in dp
        local_solver sparse
        coarse_solver sparse
        grbm_tol 1e-6
        prt_summary yes
        prt_rbm yes
        prt_debug 3
        corner_dimensionality 6     // for dp only
        corner_algorithm 3          // for dp only
        corner_augmentation none    // for dp only
  END
```

### 2.4.1 Corner Algorithms

Corner selection is an important issue (and an ongoing research area) for **FETI-DP**. Several algorithms are available. They all vary by the total number of corners picked in the model for the coarse problem. The various algorithms are intended to give a little more power to the advanced user. The more corners that are picked, the quicker the solution will converge. The disadvantage being that there might not be enough memory available for these corners, hence, **Salinas** might abort because of this memory depletion. Memory statistics can be observed and with experience, the advanced user can pick the optimal corner algorithm. The possible choices for the various parameters are given in Table 7. All the options for each corner parameter are listed such that the first option for each parameter picks the least amount of corners.

Typically, corner algorithm 15 selects the minimal number of corner points. This is a useful option to try if memory becomes an issue when running on large numbers of processors. As noted above, smaller coarse grids increase the number of iterations to convergence.

Corner algorithm 14 selects three corners between along the interface between two neighoring subdomains ($\Gamma_{ij}$ designates the interface between subdomain $i$ and subdomain $j$). The first node is selected as the node along $\Gamma_{ij}$ that touches the most subdomains. The second node is the node that maximizes the distance between any two nodes along $\Gamma_{ij}$. The third node is selected to maximize the triangular area created by three non-colinear nodes along $\Gamma_{ij}$. Corner algorithm 14 will typically select less corner nodes than Corner algorithm 3.

Note that additional corner nodes can be placed in a special file, `extra_nodes.dat`. Nodes in this file will be added to the current corner selection algorithm. While this method is seldom useful, it can help in cases where an isolated element is causing catastrophic problems. The format of `extra_nodes.dat` is not reported here. It is identical to `extra_node.dat` which is written by feti. The usual approach is to edit this file to keep only the nodes of interest.

Table 7: Corner Options

| Corner Parameter | Corner Option | Description |
|---|---|---|
| Corner Algorithm | 0 | Picks 1 corner per interface |
| Corner Algorithm | 1 | Most robust algorithm |
| Corner Algorithm | 2 | Picks 2 corners per interface |
| Corner Algorithm | 3 | Picks 3 corners per interface |
| Corner Algorithm | 9 | Picks all interface nodes *debug only* |
| Corner Algorithm | 14 | Improved version of Corner Algorithm 3 |
| Corner Algorithm | 15 | Improved version of Corner Algorithm 0 |
| Corner Algorithm | 16 | No automatic corners. (uses `extra_nodes.dat`). |
| Corner Algorithm | 17 | like 3, but add corners for conms |
| Corner Dimensionality | 3 | Fixes 3 translational d.o.f. per corner |
| Corner Dimensionality | 6 | Fixes *all* d.o.f. per corner |
| Corner Augmentation | none | no additional corners are selected |
| Corner Augmentation | edge | Additional corners on interface edges are selected. (Stiffness weighted). |
| Corner Augmentation | subdomain | Additional corners per subdomain are selected. |

### 2.4.2 Levels of Diagnostic Output

This section is under construction.

The **prt_debug** flag takes various values from 0-4. Table 8 shows the various values and their result. Note, for **prt_debug** value of 3, a file named *corner.data* is written. The format is as follows:

```
Ncorners
```

```
GlobalId LocalId SubdomainId Xpos Ypos Zpos
.
.
.
GlobalId LocalId SubdomainId Xpos Ypos Zpos
```

*Ncorners* is the total number of corners, *GlobalId* is the global id of the corner, followed by the local id (*LocalId*), the subdomain on which the corner exists (*SubdomainId*), and the coordinates of the corner (*Xpos Ypos Zpos*).

Table 8: Prt_Debug Options

| prt_debug value | Result |
|---|---|
| 0 | No Output |
| 1 | Some Output |
| 2 | Lot of Output |
| 3 | Output + Corner.data file |
| 4 | Output + Corner.data file + matlab files |

## 2.5 ECHO

Results, in ASCII format, from the various intermediate calculations may be output to a results file, e.g. *example.rslt*, where the filename is generated by taking the basename of the text input file (without the extension) and adding *.rslt* as an extension. Output to the results file is selected in the **Salinas** input file using the **ECHO** section. An example is given below, and the interpretation of these keywords is shown in Table 9.

```
echo
    materials
    elements
    jacobian
    all_jacobians
    timing
    mesh
    echo
    input
    nodes
end
```

Table 9: ECHO Section Options

| Option | Description |
|---|---|
| materials | material property info, e.g. E, G |
| elements | element block info, i.e. what material, element type, etc |
| jacobian | block summary of jacobians |
| all_jacobians | jacobians for every element |
| mesh | summary of data from the input **Exodus** file |
| echo | dumb echo of input *(for parse errors)* |
| input | summaries of many sections |
| nodes | nodal summary |
| timing | timing information |
| subdomains “0:3:6,10” | Controls which processor will output results file |
| mass | mass properties in the basic coordinate system |
| feti_input | |
| displacement | nodal displacements (better in output section) |
| velocity | nodal velocities (better in output section) |
| acceleration | nodal accelerations (better in output section) |
| force | applied forces (better in output section) |
| eforce | element force for beams |
| pressure | applied pressures (better in output section) |
| strain | element strains at centroids |
| stress | element stresses at centroids |
| vonmises | von mises stress only |
| vmrs | RMS quantities (random vibration only) |
| energy | element strain energy and strain energy density |
| genergies | global kinetic and strain energy sums |
| eorient | element orientation |
| ElemEigChecks | element eigenvalue ratios |
| all | everything |
| none | nothing |

Note that if **none** is used, the order of selection is important. Thus, if you add **none** at the end of the list, no output will be provided in the echo file. However, if you put **none nodes** then only nodal summary information will be included. Entering **nodes none mesh** only outputs the mesh information (**nodes** information is cancelled by the **none**).

The mass properties may only be reported in this section (i.e. at this time there is no mass property report in the **outputs** section). The mass properties reports the total mass, the center of gravity and the moments of inertia of the system. All are reported in the basic coordinate system. Note that moments are about the origins, not about the center of gravity. Masses are reported in a unit system consistent with the input, whether or not the **WtMass** parameter has been used (see section 2.3).

In parallel calculations, one results file is written per subdomain. Only data associated with that subdomain are written to the file. Use the "subdomains" option to specify which subdomains for which data will be written. See the comments in the paragraphs below.

## 2.6 OUTPUTS

The **outputs** section determines which data will be written to selected output files. All geometry based finite element results are written to an output exodus file. The name of this file is generated by taking the base name of the input exodus geometry file, and inserting *-out* before the file extension. For example, if the input exodus file specification is *example.exo*, output will be written to *example-out.exo.* When using a multicase solution *(section 2.1.1)*, the case identifier is used in place of "out". More details are available in the **FILE** section (2.9).

Various non-geometry based finite element data, such as system matrices and tables may be available in Matlab compatible format, or in Harwell-Boeing format. These ASCII files have the *.m* or *.hb* file extensions respectively. The base file names are derived from the type of data being output. These files are generated in the current working directory.

In the following example, the mass and stiffness matrices will be output in Matlab format, but the displacement variables, stresses and strains will not be output. All the various options of the **OUTPUT** section are shown in Table 12. The next sections describe each of the options and their results assuming an input file named *example.inp* and a geometry file named *exampleg.exo*.

```
OUTPUTS
        maa
        kaa
```

```
                faa
        //      displacement
        //      stress
        //      strain
        //      energy
        END
```

### 2.6.1 Maa

Option **maa** in the **OUTPUTS** section will output the analysis-set mass matrix to a file named *example_Maa.m*. If the harwellboeing option is selected, output will also go a file named *example_Maa.hb*.

### 2.6.2 Kaa

Option **kaa** in the **OUTPUTS** section will output the analysis-set stiffness matrix to a file named *example_Kaa.m*. If the harwellboeing option is selected, output will also go a file named *example_Kaa.hb*.

### 2.6.3 Faa

Option **faa** in the **OUTPUTS** section will output the analysis-set force vector to a file named *example_Faa.m*. If the harwellboeing option is selected, output will also go a file named *example_Faa.hb*.

### 2.6.4 ElemEigChecks

Option **ElemEigChecks** will turn on the element output of the first flexible eigenvalue, the largest eigenvalue, and the ratio of the two. The output will be stored in the Exodus output file. The element variable names for the smallest flexible eigenvalue, largest eigenvalue, and ratio of the two are *elam_min*, *elam_max*, and *elam_rat*, respectively. Note: All 3-d and 2-d elements have this capability. The **Beam2**, **OBeam**, **Spring**, **Truss**, **Spring3**, and **RSpring** elements are also supported. All remaining elements will output values of zero. Finally, if *elam_max/elam_min* is greater than $10^{20}$, then the value of *elam_rat* will be set to 1e20.

### 2.6.5 Elemqualchecks

Option **Elemqualchecks** takes either one of two choices, **on** or **off**. The default is **on**. If this option is **on**, then all of the elements in the input file are checked for quality using methods developed by Knupp (Ref. 4). Knupp uses a condition number to evaluate the health of an element. The following table shows the elements currently checked and their acceptable ranges.

| Element Type | Full Range | Acceptable Range |
|---|---|---|
| Hex8 | $1-\infty$ | $1-8$ |
| Tet4 | $1-\infty$ | $1-3$ |
| Tria3 | $1-\infty$ | $1-1.3$ |
| TriaShell | $1-\infty$ | $1-1.3$ |
| Quad4 | $1-\infty$ | $1-4$ |
| Wedge6 | $1-\infty$ | $1-5$ |

If the element's condition numbers falls outside the acceptable range a warning message is printed. The value output with the warning is normalized by the maximum number of the acceptable range for that element.

In addition to these checks, solid elements are checked for negative volumes. This can occur if the node ordering for the element establishes a "height" vector using the right hand rule that is in the opposite direction of the actual element height. In other words, the nodes should normally be ordered in a counter clockwise direction on the bottom surface of the element. Some codes such as Nastran, are insensitive to this ordering. If element checks are run, then Salinas will correct (and report) any solid elements found to have negative volumes. Without these corrections, the code will continue, but results that depend on these elements are suspect.

It is strongly recommended that any exodus file with negative volumes be corrected.

### 2.6.6 Displacement

Option **disp** in the **OUTPUTS** section will output the displacements calculated at the nodes to the output exodus file.

### 2.6.7 Velocity

Option **velocity** in the **OUTPUTS** section will output the velocities at the nodes to the output exodus file.

### 2.6.8 Acceleration

Option **acceleration** in the **OUTPUTS** section will output the accelerations at the nodes to the output exodus file.

### 2.6.9 Strain

Option **strain** in the **OUTPUTS** section will output the strains for all the elements to the output exodus file. For more information on stress/strain recovery, see section 4.

### 2.6.10 Stress

Option **stress** in the **OUTPUTS** section will output the stresses for all the elements to the output exodus file. For more information on stress/strain recovery, see section 4.

### 2.6.11 VonMises

Option **VonMises** in the **OUTPUTS** section will output the von Mises stress for all the elements to the output exodus file. For volume elements, the output will be the von Mises stress of the element. Surface elements define stresses on the top, center and bottom layers. The output will be the maximum of these 3 values.

Note that the von Mises stress is computed and output as a portion of the output if full stress recovery is requested. This option provides a mechanism for reducing output. Thus, if full stress output is requested, then the **VonMises** will provide no additional output. In other words, specifying both **VonMises** and **stress** in the outputs section is redundant, but does not result in an error.

### 2.6.12 VRMS

Option **vrms** will output computed root mean squared (RMS) quantities from a random vibration analysis. These quantities are written to a separate output file. Quantities output include the RMS displacement, acceleration and von Mises stress. In addition, the $D$ matrix terms which contribute to the von Mises stress are output[9].

### 2.6.13 Energy

Option **energy** in the **OUTPUTS** section will place strain energies and strain energy density in the output exodus file. Note that the current implementation of strain energies requires recomputation of the element stiffness matrix, which can be expensive.

[9]For a definition of $D$, see Reese, Field and Segalman, *A Tutorial on Design Analysis Using von Mises Stress in Random Vibration Environments* Shock and Vibr. Digest, Vol. 32, No. 6, Nov 2000.

### 2.6.14 GEnergies

Option **GEnergies** in the **ECHO** or **OUTPUTS** section will trigger computation of global energy sums for the results and output exodus file, respectively. For the **ECHO** case, the computation includes the following.

**strain energy** The strain energy is computed from $u^T Ku/2$ where $u$ is the displacement and $K$ is the current estimate of the tangent stiffness matrix. Note that this may not be complete for nonlinear solutions. Linear viscoelastic materials have contributions that will not be included in this sum.

**kinetic energy** Computed as $v^T Mv/2$. Here $v$ is the velocity and $M$ is the mass matrix.

**work** The **work** is defined as,

$$W(t) = \int_{x(0)}^{x(t)} F(x)dx$$

where $F$ is the force and $dx$ is the distance traveled. This can be restated as an integral over time.

$$W(t) = \int_0^t F(\tau)v(\tau)d\tau$$

where $v = dx/dt$ is the velocity. We approximate this at discretized time $t_n$ as,

$$W_n \approx \sum_i^n F_i v_i \Delta t$$

Note that this is a sum over time using the simplest method possible. Because of integration error, it may not be completely consistent with the other energies above. For the **OUTPUTS** case, the total energy is written out at each time step.

### 2.6.15 Nodalstrain

Currently, **nodalstrain** option is not implemented.

### 2.6.16 Nodalstress

Currently, **nodalstress** option is not implemented.

### 2.6.17 Harwellboeing

Option **harwellboeing** in the **OUTPUTS** section will output the mass and stiffness matrices in Harwell-Boeing format to files with *.hb* extension.

### 2.6.18 Mfile

Option **mfile** will cause **Salinas** to output various Mfiles like *Ksrr.m*, *Mssr.m*, etc. These files are mainly used by the **Salinas** developers for code maintenance and verification. Since many of these files can be quite large, caution should be exercised when using this option on large models.

### 2.6.19 Force

Option **force** in the **OUTPUTS** section will output the applied force vector to the output exodus file.

### 2.6.20 EForce

Option **eforce** in the **OUTPUTS** section will output the element forces for beams and springs to the output exodus file. Each two dimensional element will have 3 force entries for each node, for a total of 6 element forces per element.

### 2.6.21 EOrient

Option **eorient** in the **OUTPUTS** section will output the element orientation vectors for all elements. The element orientation is a design quantity that normally does not change significantly through the course of an analysis. This output is provided to help in model construction and debugging.

The orientation vectors are output as nine variables that collectively make up the three vectors required for element orientation. The output variables and the associated meanings for various elements are shown in tables 10 and 11 respectively.

### 2.6.22 Pressure

Option **pressure** in the **OUTPUTS** section will output the applied pressure to the output exodus file as an element variable. Note that there is no output for different sides of an element. Thus, if there is pressure applied to more than one face of an element, the output will represent only one of these pressures. Also note

Table 10: Element Orientation Outputs

| **Name** | **Description** |
|---|---|
| EOrient1-X | first orientation vector |
| EOrient1-Y | |
| EOrient1-Z | |
| EOrient2-X | second orientation vector |
| EOrient2-Y | |
| EOrient2-Z | |
| EOrient3-X | third orientation vector |
| EOrient3-Y | |
| EOrient3-Z | |

Table 11: Element Orientation Interpretation

| **Element** | **EOrient1** | **EOrient2** | **EOrient3** |
|---|---|---|---|
| Beam2 | axial | first bending (I1) | 2nd bending (I2) |
| Shells | Element X | Element Y | Normal |
| Solids | Element X | Element Y | Element Z |
| HexShell | Element X | Element Y | thickness |
| ConM | NULL | NULL | NULL |

that the output provides a single pressure variable per element, and is not directly related to a particular element face. For most applications this provides a useful tool for checking input loads.

Table 12: OUTPUT Section Options

| Option | Description |
|---|---|
| maa | mass matrix in the a-set |
| kaa | stiffness matrix in the a-set |
| faa | force vector in the a-set |
| elemqualchecks on \|\| off | default is on |
| ElemEigChecks | outputs first flexible eigenvalue, largest eigenvalue, and the ratio of the two for each element |
| disp | displacements at nodes |
| velocity | velocity at nodes |
| acceleration | acceleration at nodes |
| strain | strain of element |
| stress | stress of element |
| vonmises | vonmises stress on element |
| vrms | RMS quantities (random vibration only) |
| energy | element strain energy and strain energy density |
| genergies | global sum of energies |
| *nodalstrain | strains at the nodes of the f.e.m. |
| *nodalstress | stresses at the nodes of the f.e.m. |
| harwellboeing | mass and stiffness matrices in Harwell-Boeing format |
| mfile | Outputs various Mfiles ( mainly for developers ) |
| locations | Outputs nodal coordinates and DOF to node map |
| force | Outputs RHS of system of equations to be solved |
| pressure | Outputs pressure load vector |
| eforce | Outputs element forces for beams |
| eorient | Outputs element orientation vectors |

*Currently, nodalstrain and nodalstress are not implemented.

## 2.7 HISTORY

All the results from the "OUTPUT" section can be output to a limited portion of the model using a history file. Only those outputs described in Table 12 are supported. Note that if the output is also specified in the OUTPUT section, there is little need to write the data in the history file. The following output section options are ignored in the history section because all history file output will be in the exodus format.

- mfile
- harwellboeing
- kaa, maa, faa
- vrms

In addition to the **output** selection options, the history file section contains information about the regions of output. The default is NO output selection. Selection may be for node sets or element blocks. Side sets may also be selected, but the side set selection is for the nodes associated with that side set, not for the elements themselves. All nodal variables selected in the history file will be output for all selected nodes. Selecting an element block automatically selects the associated nodes in that block. The format for the selection is the same as that of the subdomains selection in the ECHO section (2.5). For example,

```
HISTORY
   nodeset '1:10,17'
   sideset  '3:88'
   nodeset '8,15' coordinate 4
   block 5,6,3
   stress
   disp
END
```

Any number of **nodeset** selections can be specified in the history section. Nodeset specifications may be followed by an optional coordinate entry. If a **coordinate** is specified, all nodal results for the nodes in the nodeset are transformed to the specified coordinate system before output to the file. If a particular node is identified in more than one specification, the last specification is used for the output. The coordinate ID of nodes in the history file may be printed out in the echo file by specifying **nodes** in the **echo** section of the input. The coordinate ID will also be written to the history file (as a nodal variable *CID*) provided any nonzero coordinate frames have been specified.

Only one **block** and one **sideset** specification is permitted in the history section. Output coordinate frames may only be specified on nodesets.

Unlike subdomains, node set and side set IDs need not be contiguous in the exodus file. The selection criteria may identify nonexistent sets. These will be silently ignored. In the above example, if the input exodus file contains no node set with ID=10, it will not be treated as an error. Node set and side set IDs in the history file will be consistent with the corresponding exodus input file.

Only one history file will be written per analysis. The name of the history file is derived from the name of the exodus output file, except that the extension is ".h".

Currently, history files in parallel can be a little confusing. In a parallel analysis, a single node may be replicated on one or more subdomains. Results from all subdomains are concatenated without sorting. As a result, a single node may be reported more than once, and the sort order of nodal results is decomposition dependent. Starting with release 1.1 element variables are sorted.

While the history file provides a convenient means for transforming coordinates, its applicability may be somewhat limited when output in many coordinate frames is desired. In particular, only a single history file is written in each analysis, and only one coordinate frame may be output per node. See the **coordinate** section (2.18) for information on obtaining the transformation matrices from each coordinate frame directly.

## 2.8 FREQUENCY

The frequency section provides information for data output from the modalFRF, directFRF, shock, modalshock, and random vibrations solution methods. One frequency file is written per analysis. The name of the frequency file is derived from the name of the exodus output file, except that the extension is ".frq". The section format follows that of the history section, except that currently the only outputs are nodal variables. Elements will be collected and placed in the geometry definition of the file, but only nodal variables are output at each frequency value. Solution methods that do not write frequency domain output silently ignore the Frequency section.

The frequency section also includes the definitions of the frequency range and step. (In the beta release notes, these were included in the solution section).

A frequency section (with some output selection region) must be selected for any solution method requiring frequency output. To fail to do so is an error, since the solution would be computed and no output provided.

```
FREQUENCY
   nodeset '1:10,17'
   sideset  '3:88'
```

```
    block 5,6,3
    disp
    acceleration
    freq_min=10     // starting frequency in HZ
    freq_step=10    // frequency increment
    freq_max=2000   // stop freq. This example has 201 frequency points.
END
```

The controls in the **frequency** section also affect data written to the results (or echo) file. In particular, the echo file contains data only for those nodes in the selection region of the **frequency** section. Selection of a specific output (such as displacement or acceleration) is independent. For example, you may echo only displacements, but write displacements and accelerations to the exodus frequency output file.

The *seacas* translator *exo2mat* may be used to translate the output into matlab format for further manipulation and plotting.

## 2.9 FILE

Disk files names are specified in the **FILE** section. The two possible parameters for the **FILE** section are,

| Option | Description |
|---|---|
| **geometry_file** | Indicates which **Exodus** file to use |
| **numraid** | Indicates how many raids are available (for parallel execution) |

In an MP environment, the file name is determined by the number of raid controllers and the processor number. The actual file name is computed by this command:

sprintf(filename,fmt, (my_proc_id%numraid)+1, my_proc_id );

where fmt is the string specified for the geometry file. The number of raid devices is defined using the keyword **numraid**. For example, on a single processor, a **FILE** section may look like this.

```
FILE
    geometry_file 'exampleg.exo'
END
```

On multiple processors this might look like:

```
FILE
  geometry_file '/pfs_grande/tmp_%.1d/junkg/datafile.par.16.%.2d'
  numraid 2
END
```

This will result in opening these files:

```
/pfs_grande/tmp_1/junkg/datafile.par.16.00
/pfs_grande/tmp_2/junkg/datafile.par.16.01
/pfs_grande/tmp_1/junkg/datafile.par.16.02
/pfs_grande/tmp_2/junkg/datafile.par.16.03
/pfs_grande/tmp_1/junkg/datafile.par.16.04
...
/pfs_grande/tmp_2/junkg/datafile.par.16.15
```

Note that if the file name is not included in quotes, it will be converted to lower case. Appendix C shows the steps involved in the parallel execution of **Salinas**.

### 2.9.1 Additional Comments About Output

A text log or *results* file can be written for the run. Details of the contents of the results file are controlled in the **ECHO** section (see section 2.5). The results file name is determined by the name of the input file, and will be in the same directory as the input text file, regardless of whether **Salinas** is being executed in serial or parallel. However, if executing in parallel, using the "subdomains" option in the **ECHO** section allows control of the number of *results* files. For example, if running on 100 processors, up to 100 result files may be output. Using **subdomains** "`0:2`" will only output three files, from subdomains 0, 1, and 2. The default is to output a *results* file only for processor zero. The results file name uses the base name of the input, with an extension of ".rslt". In a parallel computation, the results file names use the base name of the input file, followed by an underscore and the processor number, then followed by the "`.rslt`" extension.

For calculations in which geometry based output requests are included (see section 2.6), an output **Exodus** file will be created. The **Exodus** file is a binary file containing the original geometry plus any any requested output variables. The output **Exodus** file name is determined from the geometry file name. The base name of the output is taken from the geometry file by inserting the text "`-out`" just before the file name extension. The output **Exodus** file will be written to the same directory where the geometry file is stored. If executing **Salinas** on a parallel machine, the **Exodus** output files should be written to the raid disks for reasonable performance.

## 2.10 BOUNDARY

Boundary conditions are specified within the **Boundary** section. Either node sets or side sets may be used to specify boundary conditions. By default, they are specified in the basic coordinate system, but an alternate system my be specified using the "coordinate" keyword (see section 2.18). The current implementation is very inefficient if any but the basic system is used (they are treated as MPCs). The following example illustrates the method.

```
BOUNDARY
  nodeset 1
    coordinate 7  // apply in coordinate system 7
       x = 0.1    // constrain x=0.1 for all nodes in set
       y = 0      // constrain y=0 throughout nodeset 1
       RotZ = 0   // constrain the rotational dof about Z
  nodeset 2
    fixed         // constrain all structural dofs in nodeset 2
  sideset=4
    fixed         // constrain all nodes in sideset 4
END
```

The descriptors for the boundary conditions are, **X**, **Y**, **Z**, **RotX**, **RotY**, **RotZ**, and **fixed**. An optional equals sign separates each descriptor from the prescibed value. The value **fixed** implies a prescribed value of zero for all degrees of freedom. Note, that only constant prescribed displacements are allowed on nodesets and sidesets.

## 2.11 LOADS

Loading conditions are specified within the **loads** section. The following example illustrates the method.

```
LOADS
   nodeset 3
      force = 1.0 0. 0.
      scale = 1000.
      function = 2
   nodeset 5
      force = 0. -1 0
   body
```

```
        gravity
        0.0 1.0 0
        scale -32.2
     sideset 7
        pressure 15.0
     sideset 12
        traction = 100.0 20.0 0.0
  END
```

Loads may be applied to node sets, side sets or the entire body (in the case of inertial loads). Loads are applied in the global coordinate system using nodesets. Pressure loads may be applied using side sets. The pressure is always normal to the surface. All loads applications are additive.

The syntax followed is to first define the region over which the load is to be applied (either **nodeset**, **sideset** or **body**). Each such region defines a *load set.* For each such definition, one (and only one) load type may be specified. However, any region definition may be repeated so that forces and moments may be applied using the same node set. The load types are,

| Option | Parameters |
|---|---|
| **force** | *val1 val2 val3* |
| **moment** | *val1 val2 val3* |
| **gravity** | *val1 val2 val3* |
| **pressure** | *val1* |
| **traction** | *val1 val2 val3* |

Following the definition of the load type, a vector (or scalar in the case of pressure loads) must be specified. The total force is the product of the load vector, the scale factor, and the nodeset distribution factor found in the exodus file. Note that in some cases the nodeset distribution factor may be zero. In that case, the total applied force will also be zero. The **pressure** loading may only be applied to side sets. The total pressure is the product of the scale factor, pressure (scalar) and sideset distribution factors. If the pressure loading in NOT normal to the sideset, the **traction** capability should be used. NOTE: Pressure will act on a surface in a compressive sense, while a traction can be specified as any vector which will act on the **sideset** specified in the direction given by the triple values specified after **traction**. Also, traction loads are applied on the faces of the shell elements in a piecewise manner, i.e., the traction load acting on a face of the element is assumed constant. If the distribution factors on the nodes of the element vary, the average of the load ( element per element ) is assumed.

NOTE: If a function is defined for a particular load, then it is assumed to be a transient load. If there is no function defined then it is assumed to be a static load. The solution procedure chosen will only use the loads that are applicable, i.e., a static solution will only use static loads and a transient solution will only use transient loads.

### 2.11.1 Consistent Loads

The loads for all of the 3-D and 2-D elements are calculated in a consistent fashion when a pressure load is applied. For more details on the implementation, see the programmer's notes.

### 2.11.2 Time Varying Loads

Additional options provide the capability of varying the load over time. The **LOADS** options include,

- **scale** with one parameter provides a scale factor to be applied to the entire load set. Only one scale may be provided per load set.
- **function**. A time varying function may be applied by specifying a function ID. Only one function may be applied per load set. The function is defined in the **FUNCTION** section (see section 2.19 on page 62). The loads applied at time $t$ for a particular load set will be the sum of the force or moment vectors summed over the nodes of the region and multiplied by the scale value and the value of the time function at time $t$.

Variation of the load over space is accomplished using node set or side set distribution factors. If these are provided in the **Exodus** file, the load set is spatially multiplied by these factors. The total loading is the sum of the loads for each load set summed over all the load set regions.

## 2.12 Load

Loading conditions *for all multicase solutions* are specified within the **load** section. See paragraph 2.1.1 for information on specifications for multicase solutions. The **LOAD** section is *identical* to the **LOADS** section described in the previous paragraph (2.11), except the the section begins with the **load**, and a load step identifier is required. The following example illustrates the required input.

```
LOAD=57
   nodeset 3
      force = 1.0 0. 0.
      scale = 1000.
      function = 2
   nodeset 5
      force = 0. -1 0
END
```

Unlike the **Loads** section, there may be multiple **load** sections in the file, with each entry corresponding to an applicable step in the solution.

## 2.13 RanLoads

The **RANLOADS** section is used to provide input information for spectral input to a random vibration analysis. Note that this input will contain both a spatial and temporal component. The **ranloads** section contains the following required keywords.

| Parameter | Argument | Description |
|---|---|---|
| **matrix** | *Integer* | matrix-function identifier |
| **load** | *Integer* | row/column identifier |

The **matrix** keyword identifies the appropriate **matrix-function** (see section 2.20). The matrix-function determines the dimensionality of the input (using the **dimension** keyword). It also determines the spectral characteristics of the load.

The spatial characteristics are determined in **load** sections within the **ranloads** definition. There must be exactly as many **load** sections as the dimensionality of input. For example, if the $S_{FF}$ matrix is a 3x3, then there should be 3 separate load sections. Each load section within the **ranloads** block must be followed by an integer indicating to which row/column it corresponds. The details of each **load** section are identical to the over all **loads** section (see 2.11) except that no time/frequency function is allowed. Note that only one load is required per row of the $S_{FF}$ matrix, but each *entry* of the matrix may have a spectral definition (identified by a real and/or imaginary **function**).

The following example illustrates the definition of a single input specification. The loading is scaled so that a 1000 lb mass located on the input point (in nodeset 12 here) is scaled to produce a unit $g^2/Hz$ loading.

```
RANLOADS
```

```
      matrix=1
      load=1
        nodeset 12
           force=0 1 0
           scale 1.00e3  // needed to convert to g
           // loads input in lbs. The PSD is in g^2/Hz.
           // F = accel * mass
           //   = accel * (scale_factor)
           //   = accel * ((1000*.00259)*384.6)
   END
```

## 2.14 Contact Data

Contact surfaces provide the ability to model structures that may be in contact part of time. Examples include a tire rolling on the road, rattling in a joint and structures under impact. Note that a tied surface capability is available for the apecial case where the surfaces will always be in contact (see section 2.15). Contact surfaces are inherently nonlinear. The simplest of such structures are characterized by frictionless contact, i.e. a restorative force is provided only in the normal direction, and slippage occurs tangent to the normal. This is the only type of contact currently supported in Salinas, though frictional and sticking contact are under development.

Contact is specified as shown in the example below.

```
CONTACT DATA
    Surface 33,17
    Friction Static = 0.0
    Search Tolerance = 1e-8
END
```

The example above defines a region of contact between sidesets 33 and 17. While contact is defined here between sidesets, it is currently limited to node/node contact. The coefficient of static friction is defined to be zero (the default). The search tolerance sets the radius for search for the nodes on the surface. The relevant parameters for contact are shown in Table 13.

The only method currently supported is node/node which is specified (perhaps confusingly) by the keyword "Surface". Also, the only supported friction model is none, specified by a coefficient of static friction of zero. Contact is enforced within the solver, and is currently supported only when using the FETI solver. Enforcement does not truly require distinguishing between the master and slave surface. The contact is enforced symmetrically. We have adjusted the input to be as consistent as possible with other Sandia codes.

Table 13: Contact Data Parameters

| Parameter | type | description |
|---|---|---|
| Surface | integer pair | master and slave sideset separated by comma or space |
| Friction Static | Real | coefficient of static friction (defaults to zero) |
| Search Tolerance | Real | Radius of search for paired nodes defaults to 1e-8 |

## 2.15 Tied Surfaces

Tied surfaces provide a mechanism to connect surfaces in a mesh that will always be in contact. Because the surfaces are always tied, the constraints may be represented by a set of linear multipoint constraints (see section 3.25). Tied surfaces are also known in the literature as “glued surfaces” or as “tied contact”. They are used almost exclusively to combine two surfaces of a mesh that have not been meshed consistently.

There are a number of ways of combining surfaces that have not been consistently meshed. The simplest method (and the only one currently under development in Salinas) constrains the nodes of the slave surface to lie on the master surface. In this method, the constraint is called *inconsistent* because the mesh does not ensure that linear stress will be maintained across the boundary. The stress and strain in the region of the constraint will be wrong. However, loads are properly transferred across the boundaries, so a few element diameters away from the boundary, the stresses and strains should be approximately correct.

In the future, the *inconsistent* tied surface will be transitioned into a fully consistent algorithm. No change in the input is expected to achieve this. It is achieved by a modification of the element stiffness matrices on the boundary.

Tied surfaces are specified by a listing of master and slave side sets. Any number of tied surfaces may be specified in the input, i.e. more than one tied surface section may occur in the input. Each tied surface section represents a *single* logical pairing of constraint side sets.

```
TIED DATA
    Surface 12, 18
    search tolerance = 1e-7
END
```

In the example above, sideset 12 is defined as a master surface. Side set 18 is the slave surface.

The relevant parameters for tied surfaces are shown in Table 14.

Table 14: Tied Surface Parameters

| Parameter | type | description |
|---|---|---|
| Surface | integer pair | master and slave sideset separated by comma or space |
| Search Tolerance | Real | Radius of search for paired nodes defaults to 1e-8 |

## 2.16 BLOCK

Each element block in the **Exodus** file, must have a corresponding **BLOCK** entry in the input file. This section contains information about the properties of the elements within the block. These properties depend on the element type. Clearly shells will require a thickness, while it is meaningless for solids. An example is provided below.

```
// the following element block is Tria3
BLOCK 32
    material 2
    tria3
    thickness 0.01
END

// the following element block is hex.
// exodus tells us it is an 8-node hex.
// The default integration mode is "UNDER"
// The only required argument is the material card
BLOCK 34
        material 3
END

BLOCK 3
    Coordinate 1
    Spring
    Kx=1e6
    Ky=0
    Kz=0
END
```

A list of the applicable attributes for different element types is shown in Table 15. Each element type is outlined in section 3.

In addition, the reference coordinate system may be defined in a block. This definition applies to all the elements of the block and the associated materials. At this point, the coordinate system is only recognized for a subset of the elements (solid elements and springs). Further information on coordinate systems may be found in section 2.18.

Table 15: Element Attributes

| Element Type | attr | keyword | Description |
|---|---|---|---|
| ConMass | 1 | Mass | concentrated mass |
| | 2 | Ixx | xx moment of inertia |
| | 3 | Iyy | yy moment of inertia |
| | 4 | Izz | zz moment of inertia |
| | 5 | Ixy | xy moment of inertia |
| | 6 | Ixz | xz moment of inertia |
| | 7 | Iyz | yz moment of inertia |
| | 8,9,10 | offset | offset from node to CG |
| Beam | 1 | Area | Area of beam |
| | 2,3,4 | Orientation | orientation vector. For the orthogonal direction |
| | 5 | I1 | First bending moment |
| | 6 | I2 | Second bending moment |
| | 7 | J | Torsion moment |
| | 9,10,11 | offset | beam offset |
| Spring | 1 | Kx | spring constant in X |
| | 2 | Ky | spring constant in Y |
| | 3 | Kz | spring constant in Z |
| Triangle | 1 | thickness | thickness |
| | 2 | offset | shell offset in normal direction |
| Quad | 1 | thickness | thickness |
| | 2 | offset | shell offset in normal direction |

## 2.17 MATERIAL

Most element blocks must specify a material. Details of that material are included in the material section. The material section contains a material identifier (which is usually an integer, but may be any string), an optional **name** keyword followed by a material name, a material type keyword and the necessary parameters. The different material types and their parameters are summarized in Table 17.

For example,

```
MATERIAL 3
    isotropic
    name "steel"
    E 30e6
    nu .3
END
```

Deterministic materials may be input as **isotropic**, **orthotropic**, **orthotropic_prop**, **anisotropic**, or **isotropic_viscoelastic**. In addition, stochastic isotropic materials may be specified as **S_isotropic**.

### 2.17.1 Isotropic Material

Isotropic materials require specification of two of the following parameters.

| Parameter | Description |
|---|---|
| **E** | Young's Modulus |
| **nu** | Poisson's Ratio |
| **G** | Shear Modulus |
| **K** | Bulk Modulus |

Isotropic materials are the default, and the keyword **isotropic** is not required.

### 2.17.2 Anisotropic Material

Anisotropic materials require specification of a 21 element $C_{ij}$ matrix corresponding to the upper triangle of the 6x6 stiffness matrix. Data is input in the order $C_{11}$,$C_{12}$, $C_{13}$, $C_{14}$, $C_{15}$, $C_{16}$, $C_{22}$, etc. The $C_{ij}$ must be preceded by the keyword **Cij**. The keyword **Anisotropic** is also required. Materials are specified in the order $xx$, $yy$, $zz$, $zy$, $zx$, $xy$. Note that this ordering varies in the literature. It differs from the ordering in Nastran and Abaqus, but is consistent with much of the published materials science data. An example input file with an anisotropic material is found in section A.2.

### 2.17.3 Orthotropic Material

Orthotropic material entry is identical to the anisotropic case with the exception that the keyword **orthotropic** replaces **anisotropic**, and only 9 $C_{ij}$ entries are specified. These entries correspond to $C_{11}$, $C_{12}$, $C_{13}$, $C_{22}$, $C_{23}$, $C_{33}$, $C_{44}$, $C_{55}$ and $C_{66}$. Like the anisotropic material definition, the order is $xx$, $yy$, $zz$, $zy$, $zx$, $xy$. Alternatively, an orthotropic material may be specified using **orthotropic_prop** and the material parameters `E1, E2, E3, nu23, nu13, nu12, G23, G13`, and `G12`.

If sensitivity analysis is being performed (see section 2.21), one indicates the parameters for analysis by following these parameters with the `+/-` characters. In the first entry method, a sensitivity analysis must be performed on all 9 parameters. In the second, each individual parameter must be requested individually. The concept is that the sensitivity is performed with respect to the labelled parameters, i.e. either the set of $C_{ij}$ parameters, or each individually labelled `E1` term.

### 2.17.4 Stochastic Material

For stochastic materials, all material properties are determined by a table look-up, based on the element ID. The file name for the table lookup is taken from the **name** identifier. The file is a standard text file with the first column corresponding to the element ID. The second column is the bulk modulus, $K$, and the third (and final) column is the shear modulus, $G$. The element IDs in the file need not be continuous, but they must be sorted in increasing order. Thus the **S_isotropic** data lookup file contains the element ID, the bulk modulus and the shear modulus, with one line for each element. The stochastic material model is very preliminary and is expected to change significantly in the next few years.

### 2.17.5 Linear Viscoelastic Material

Linear viscoelastic materials require the specification of the density, and the limiting moduli E_g, E_inf, G_g, G_inf. The subscript 'g' refers to the glassy modulus, which occurs at $t = 0$, or $\omega = \infty$. The subscript 'inf' refers to the rubbery modulus, which occurs at $t = \infty$, or $\omega = 0$. In addition the Prony series for the viscoelastic materials have to be specified using keywords K_coeff, K_relax, G_coeff, and G_relax. All of these parameters are required.

For the bulk modulus $K$, the Prony series parameters are defined by the following equation:

$$K(t) = K_{inf} + (K_g - K_{inf}) \sum_i K_{coeff}[i] * e^{-\frac{t}{K_{relax}[i]}} \tag{11}$$

A similar equation holds for the shear modulus. Note that, the K_coeff and G_coeff MUST sum to 1.0 (individually). Otherwise, the formulation is inconsistent. That is,

$$\sum_i K_{coeff}[i] = \sum_i G_{coeff}[i] = 1.0 \tag{12}$$

Note that the number of terms in K_coeff and K_relax must be the same, and the number of terms in the G_coeff and G_relax must be the same. However, the number of terms in the K series does not have to equal the number of terms in the G series. Thus, one could simulate a case where the material shear modulus G is viscoelastic, but the bulk modulus is not. In this case, the latter would have no terms in its series.

Optional parameters for viscoelastic materials include reference (T_0) and current temperature (T_current), and the WLF constants C_1 and C_2. (more explanation of the Williams-Landel-Ferry (WLF) equation is given below). Also, two constants may be specified that describe the curve fit for the shift function, a_T1 and a_T2, in the case when T_current - T_0 is negative. The equation was provided by Terry Hinnerichs and is a good characterization of many viscoelastic materials. Its form is

$$a\_T = a\_T1 * (1 - e^{a\_T2*(T\_current - T\_0)}) \tag{13}$$

If these optional parameters are not specified, default values are used, as shown in the table below. Note that equation 13 will only be used to compute the shift

Table 16: Default Parameters for Viscoelastic Materials

| parameter | default value |
|---|---|
| T_0 | 0.0 |
| T_current | 0.0 |
| C_1 | 15.0 |
| C_2 | 35.0 |
| aT_1 | 6.0 |
| aT_2 | .0614 |

functions if the parameters aT_1 and aT_2 are specified. Otherwise, the standard WLF equation is used, as described below.

If the parameters aT_1 and aT_2 are not specified, then the shift factors are computed using the WLF equation. This equation is frequently used to determine an approximate set of shift factors when experimental data for a particular material is not at hand. The shift factors computed from this equation are used to scale the coeffecients in the Prony series. The shift factors computed from the WLF equation are a strong function of temperature. The WLF equation is as follows

$$log(a_T) = -\frac{C\_1(T\_current - T\_0)}{C\_2 + T\_current - T\_0} \tag{14}$$

where T_current is the current temperature in the block, and T_0, C_1, and C_2 are material parameters that are determined experimentally. If C_1 and C_2 are not known for a particular material, then the default alues given above are typically used. Typically, T_0 is the glass transition temperature of the material of interest. More explanation of the WLF equation can be found in the books by Aklonis,[5] and Ferry.[6]

After computing the shift factors using one of the two approaches given above, the relaxation times are shifted. This occurs before computations begin, using the relations

$$G_{coeff}[i] = a_T G_{coeff}[i] \tag{15}$$
$$G_{coeff}[i] = a_T G_{coeff}[i] \tag{16}$$
$$\tag{17}$$

These shifts are automatically computed given T_0, T_current, C_1, and C_2, so that the user does not need to shift the relaxation times beforehand. Note that if these parameters are not specified in the input file, then they are given default values that result in no shifting of relaxation times. In such a case, $a_T = 1$.

An example material block for a linear viscoelastic material looks like:

```
MATERIAL 9
 isotropic_viscoelastic
 name "foam"
 T_0=0
 T_current=25
 C_1=1
 C_2=2
 aT_1=6.0
```

```
     aT_2=.06
     K_g 30.0e6
     K_inf 10.0e6
     G_g 10.0e1
     G_inf 12.0
     K_coeff .5 .5
     K_relax 3.0 2
     G_coeff .5 .5
     G_relax 1 3
     density 0.288
    END
```

Note that the coefficients of both $K$ and $G$ sum to 1.0. This is necessary for a consistent formulation.

A note on viscoelastic materials: currently there are two time integration algorithms available in Salinas, the Newmark-beta method, and the generalized alpha method. At this time viscoelastic materials only work with the Newmark beta method.

### 2.17.6 Density

For solutions requiring a mass matrix, all material specifications require a keyword **density** followed by a scalar *value*.

Table 17: Material Stiffness Parameters

| material type | parameters |
|---|---|
| isotropic | any two of $K$, $G$, $E$ or $\nu$ |
| orthotropic | nine $C_{ij}$ entries |
| orthotropic_prop | $E1$, $E2$, $E3$, $nu23$, $nu13$, $nu12$, $G23$, $G13$, $G12$ |
| anisotropic | 21 $C_{ij}$ entries |
| S_isotropic | file containing $K$ and $G$ |

## 2.18 COORDINATE

Coordinate systems may be defined for reference to the materials and boundary conditions. As reported in the "history" section, nodal results may also be reported in arbitrary coordinate frames in the history file only (see section 2.7). Note that all nodal locations, outputs, etc. are always defined in the basic coordinate system in the standard exodus files. These new coordinate systems are always defined based on three *locations*, which are defined in the basic coordinate system. These locations are illustrated in Figure 1.

1. The location the origin of the new coordinate system, $v_1$.
2. A point on the $Z$ axis of the new system. This is illustrated in the figure by the vector $v_2$. Note however, that the location is required, which is the vector sum of $v_1 + v_2$.
3. A point in the $\tilde{X}\tilde{Z}$ plane of the new system, illustrated by the vector $v_3$. Note that vector $v_3$ need not be orthogonal to $v_2$, but it may not be parallel to it.

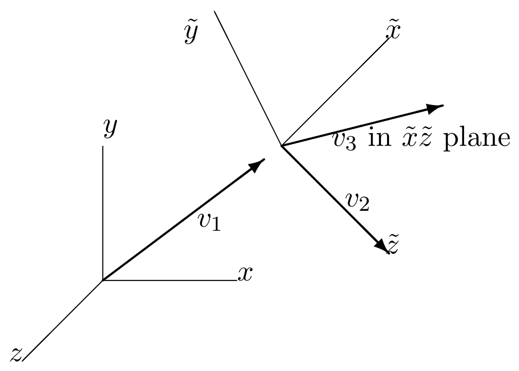

Figure 1: Coordinate System Definition Vectors

Coordinate systems for cartesian, cylindrical and spherical coordinates may be defined. In the case of noncartesian systems, the $XZ$ plane is used for defining the origin of the $\theta$ direction only.

This example creates a cylindrical system located at a point (1,1,1) with the cylindrical axis in the (0,0,1) direction and the radial coordinate in the global $Y$ direction.

```
Coordinate 7
    cylindrical
    1 1 1
    1 1 2
    1 2 1
```

```
END
```

The keywords for the coordinate system definitions are:

1. RECTANGULAR or CARTESIAN to define a cartesian system,
2. CYLINDRICAL for a cylindrical, i.e. polar system, and
3. SPHERICAL for a spherical system.

If "input" is selected in the ECHO section then the transformation matrix will be output in the `.rslt` file (section 2.5). The transformation matrix is a unitary matrix which can be used to transform vectors from one system to another. If we let $T$ be the matrix reported in the `.rslt` file, then the transformation from the basic system to the rotated frame is given by,

$$v_{new} = T^T v_{basic}$$

|  |  |  |
|---|---|---|
| where | $v_{new}$ | is the vector in the new coordinates, |
|  | $v_{basic}$ | is the vector in the basic system, and |
|  | $T^T$ | is the transpose of the `.rslt` matrix reported. |

While the history file provides a convenient means for transforming coordinates, its applicability may be somewhat limited. In particular, only a single history file is written in each analysis, and only one coordinate frame may be output per node (see section 2.7).

## 2.19 FUNCTION

Time or frequency dependent functions for transient and frequency response analysis can be defined using the **function** section. The following examples illustrate the use of this section.

```
FUNCTION 1
  type LINEAR
  name "test_func1"
  data 0.0 0.0
  data 0.0150 0.0
  data 0.0152 1.0
  data 0.030 0.0
END
```

```
FUNCTION 2
// This is a smooth pulse with time duration .05
//  it peaks at approximately t=.02 sec with a
//  value of 0.945.
// The equation is y(t)=-800*t^2 + 8.9943*sqrt(t)

  type POLYNOMIAL
  name "poly_fun"
  data 0. 0.
  data 2.0 -8.0e2
  data 0.5 8.9443
END
```

The keywords for the function definitions are:

1. TYPE to define the functional form,
2. NAME for reference in echo and output, and
3. DATA for the functional parameters.

Currently there are two types of functional forms, **linear** and **polynomial**. The data elements are defined in the context of this form.

### 2.19.1 Linear Functions

For linear functions, the data elements are points of the function where the user defines the value of the independent variable (e.g. time) and the corresponding value of the function. Linear interpolation is used to find all other values of the function. In order to make the linear interpolation unique, the order of the input data is important. Input checks will ensure that time on subsequent data points is always greater than or equal to time on the previous data point so that curves cannot double back on themselves. For example,

```
FUNCTION 3
    name "illegal_fun"
    type linear
    data 0.00 0.
    data 0.01 1.
    data 0.05 1.
    data 0.04 0.   //illegal. the first column must never decrease
END
```

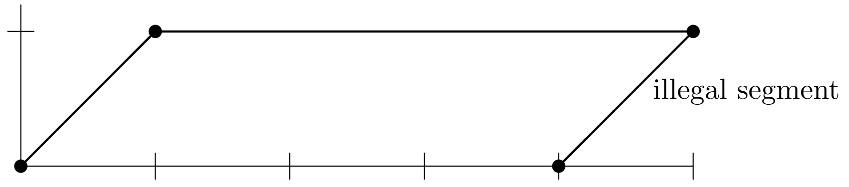

Figure 2: Linear function #3. "illegal_fun"

Linear functions will extrapolate by using the value of the nearest data point. For example, in the following function, f(t=0.3) = 0.5.

```
FUNCTION 5
    name "extrap_fun"
    type linear
    data 0.00 0.
    data 0.01 1.
    data 0.02 0.5
END
```

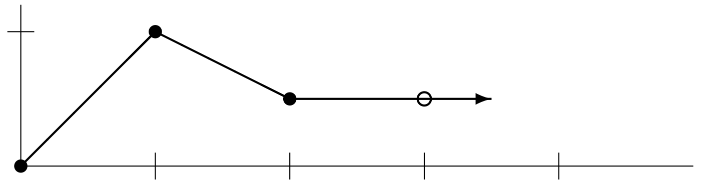

Figure 3: Linear function #5. "extrap_fun"

### 2.19.2 Polynomial Functions

For polynomials, the data points given are the exponent of the independent variable and a scale factor for that term. The independent variable taken to any real power will always be evaluated as positive. If powers are repeated, their coefficients will sum. For example,

```
FUNCTION 6
    name "poly_fun"
    type polynomial
    data 0.0 0.
    data 1.0 1.
    data 2.0 0.1
    data 1.0 0.5
END
```

is equivalent to

```
FUNCTION 6
    name "poly_fun"
    type polynomial
    data 0.0 0.
    data 1.0 1.5
    data 2.0 0.1
END
```

The function value as a function of the independent variable $t$ is,

$$f(t) = 1.5t + 0.1t^2.$$

## 2.20 MATRIX-FUNCTION

This section provides for input of a matrix function as is used in a cross correlation matrix for input to a random vibration analysis. In the limit of a single input these reduce to a single function (as described in the previous section). Note that a matrix-function can have arbitrary symmetry and can be complex. An important feature of the matrix-function is that each entry of the matrix is a function of frequency (or time).

The Matrix-Function is illustrated in the following example.

```
MATRIX-FUNCTION 1
   name 'cross-spectral density'
   symmetry=hermitian
   dimension=2x2
   nominalt=20.1
   data 1,1
```

```
      real function 1 scale 1.0
    data 1,2
      real function 12
      imag function 121 scale -3.0
    data 2,2
      real function 22 scale 0.5
  END
```

Matrix functions have the following parameters.

**NAME** allows you to optionally enter a string by which the matrix-function will be identified in subsequent messages.

**SYMMETRY** identifies the matrix symmetry. Options are “none”, “symmetric”, “asymmetric” and “hermitian”. If the matrix is not square, only “none” can apply. The default for this optional parameter is “symmetry=none”.

**DIMENSION** specifies the dimension of the matrix. If not specified, it defaults to 1x1. The dimension is specified as the number of rows, an “x” and the number of columns. No space should be entered between the terms.

**DATA** Each data entry specifies one entry in the matrix-function. The *data* entry must be immediately followed by the matrix location specified as a row, column pair. Again, no spaces may be inserted in the location entry. The *data* parameters uses two keywords.

- “real” identifies the real component of the entry. It must be followed by a function reference, and optionally by a scale factor.
- “imag” identifies the imaginary component of the entry. It must be followed by a function definition, and an optional scale factor.

**NOMINALT** Used only for echoing the matrix values. If *input* is specified as an *Echo* option (see section 2.5) general information from the matrix function are written to the log file (the *.rslt* file). If, a *nominalt* entry also exists, then the matrix entries are written for that nominal time (or frequency). Only one such output can be specified. It provides a means of checking the input to assure the matrix values are correct at a single time (or frequency) value.

## 2.21 SENSITIVITY

This section controls global parameters related to sensitivity analysis. Sensitivity analysis is not performed in **Salinas** unless this section is present in the input file. The following example illustrates the legal keywords.

```
SENSITIVITY
  values all
  vectors 1 thru 3  5  7 thru 9
  iterations 8
  tolerance 1e-7
END
```

The keywords **values** and **vectors** are used to control what types of sensitivities are computed for which cases in the analysis. In modal analysis, these refer to the eigenvalues and eigenvectors, respectively, and the case numbers represent the mode numbers. In static and transient analysis, **vectors** refers to the displacement vector results, and **values** has no meaning. Also, in modal analysis, eigenvalue sensitivities are always computed when eigenvector sensitivities are requested for a mode. Allowable values are:

```
vectors all              // compute for all cases/modes
vectors none             // compute for no cases/modes
vectors                  // default, same as all
vectors 1 2 3 5          // cases/modes 1,2,3,5
vectors 1 thru 3 5       // using thru to define range
```

Omitting the keyword **vectors** (or **values**) is equivalent to not requesting those sensitivities; in other words, it is equivalent to **vectors none**. The keywords **iterations** and **tolerance** are used in computing eigenvector derivatives. The default values are `10` and `1.0e-06`, respectively.

Sensitivity results are scaled by multiplying the derivative with respect to a parameter by the nominal value of that parameter. In this was, the units of the sensitivity coefficients are the same as the units of the nominal response results. Furthermore, in order to determine the absolute change in a response resulting from a relative change in a parameter, simply multiply the sensitivity of the response with respect to that parameter by the relative change. For example, multiply by 0.10 for the effect of a 10% change in the parameter.

Sensitivity results are output to the same file as the nominal results. The arrangement of the output varies depending on the analysis. For statics, the nominal result is output, followed by the sensitivity result for each parameter. For eigenanalysis, the nominal frequencies and eigenvectors are output, followed by the eigenvalue and eigenvector sensitivities with respect to the first parameter, the second parameter, and so on. The eigenvalue sensitivities are placed in the time field of each output record, just as the frequencies are for the nominal modal parameters. For transient analysis, the nominal response for each time step is output, followed by the sensitivities for that time step. Then the nominal results for the next time step are output, and so on.

The selection of parameters is controlled by the inclusion of a +/- symbol following a parameter in the input deck. Examples of valid sensitivity parameter definitions are:

```
MATERIAL 1
  E 10e6 +/- 1e6              // absolute tolerance specified
  density 2.59e-4 +/-         // no tolerance, use default
END

BLOCK 1
  area 0.10 +/- 5 %           // relative tolerance specified
END

BLOCK 2
  thickness  +/- 1 %          // relative to exodus attr
END

LOADS
  nodeset 1
  force 0. 0. 1000 +/- 0 0 10  // tolerance for vector param
END
```

Note that the tolerances are specified on the parameters where they normally appear in the input file. That is, these definitions do not appear in the **SENSITIVITY** section.

## 2.22 DAMPING

This section allows input of simple global viscous damping models, using either modal damping rates or stiffness and mass proportional damping. The various options for the DAMPING section are shown in Table 18.

Table 18: DAMPING Section Options

| Parameter | Description |
|---|---|
| alpha | mass proportional damping parameter (real) |
| beta | stiffness proportional damping parameter (real) |
| gamma | uniform modal damping rate (% of critical) (real) |
| mode | individual modal damping ratio (fraction of critical) (integer, real) |
| ratiofun | index of function to define modal damping ratios |

The damping matrix or modal damping coefficient is determined by summing contributions from all damping parameters given in Table 18. For modal superposition-based analysis, including **modalfrf**, **modalranvib** and **modaltransient**, all the given parameters are defined. For direct implicit transient analysis, the modal damping parameters apply only to modes for which eigenvalues and eigenvectors have previously been computed. This depends on the presence of the keyword **nmodes** in the solution section of the input file.

The effect of the mass and stiffness proportional parameters on modal damping depends on the frequencies of the modes. For modal-based analysis, the damping rate for mode $i$ with radial frequency $\omega_i$ is given as

$$\zeta_i = \alpha/(2\omega_i) + \beta \cdot \omega_i/2 + \Gamma + \texttt{mode}[\texttt{i}] + ratiofun(i)$$

where the viscous damping term in the modal equilibrium equation is $2\zeta_i\omega_i$. For example the following damping input section could be used in a modal transient analysis.

```
DAMPING
  alpha  0.001
  beta 0.00005
  gamma 0.005
  mode  1  0.01
  mode  2  0.005
  mode  3  0.015
END
```

It produces the following damping ratios.

| Mode | modal damping ratio | modal viscous damping term |
|---|---|---|
| 1 | $0.015 + 0.001/(2\omega_1) + 0.00005\omega_1/2$ | $0.030\omega_1 + 0.001 + 0.00005\omega_1^2$ |
| 2 | $0.010 + 0.001/(2\omega_2) + 0.00005\omega_2/2$ | $0.020\omega_2 + 0.001 + 0.00005\omega_2^2$ |
| 3 | $0.020 + 0.001/(2\omega_3) + 0.00005\omega_3/2$ | $0.040\omega_3 + 0.001 + 0.00005\omega_3^2$ |

In direct (i.e. non-modal-based) transient analysis, the same damping input section would produce the same damping ratios if all the modes used in the modal transient analysis were also available for the direct transient. Conversely, if no modes were available, the above damping input section would produce a physical damping matrix $C = 0.001M + 0.00005K$.

The **ratiofun** keyword permits definition of modal damping terms based on a frequency dependent function. The associated function definition (see section 2.19) provides a table lookup for damping ratios. For example, consider a system with modes at 200 and 500 Hz. The following example will establish modal damping ratios of .03 and .06 respectively. The function describes a line defined by $ratio(f) = 0.01 + 0.1/1000f$.

```
DAMPING
   ratiofun=100
END

FUNCTION 100
   type=linear
   data 0 0.01
   data 1000 0.11
END
```

## 2.23 NOX

This section allows input of nonlinear solver options to the NOX nonlinear solver. Currently, only a small subset of the many options available in NOX and described at http://software.sandia.gov/nox are able to be specified. These include the following:

Table 19: NOX Nonlinear Solver Options

| Option | Choices |
|---|---|
| nonlinear_solver_method | trust_region_based* |
| | line_search_based (default) |
| nonlinear_direction_method | newton (default) |
| | modified_newton |
| | steepest_descent |
| | nonlinearcg |
| nonlinear_linesearch_method | full_step (default) |
| | polynomial |
| | more_thunte |
| | backtrack |
| | nonlinearcg |

*Default options for this choice are used. Specification of direction and linesearch methods apply only to line_search_based.

## 3 Element Library

Short descriptions of each of the types of elements follow. Most of the parameters for the element are supplied either in the database file (i.e. **Exodus** file) or in the text input file (*.inp). If parameters exist in both locations, the values specified in the text input will over ride the exodus database specification.

### 3.1 Hex8

The **Hex8** is a standard 8 node hexahedral element with three degrees of freedom per node. The **Hex8** element has 8 integration points. The shape functions are trilinear. It supports isotropic and anisotropic materials.

There are two variations of **Hex8**. The default element is an under integrated Hex with properties similar to those of most commercial finite element codes. The underintegration produces an element that is soft relative to a fully integrated element. It may be specified by **Hex8** or by **Hex8u**.

The fully integrated Hex is specified by **Hex8F**. While it performs adequately when the element shape is nearly cubic, it performs quite poorly for larger aspect ratios. For most problems involving bending the Hex8u is recommended.

## 3.2 Hex20

The 20 node variety of Hex element provides quadratic shape functions. It is a far better element than the Hex8, and should be used if possible. The Hex20 element in Salinas is very similar to elements found in most commercial codes.

## 3.3 Wedge6

The Wedge6 is a compatibility element for the **Hex8**, it is not recommended that the entire mesh be built of **Wedge6** elements. They are primarily intended for applications where triangles are naturally generated in mesh generation.

## 3.4 Wedge15

The Wedge15 element adds midside nodes to the Wedge6. Like the Hex20 and Tet10, it has quadratic shape functions, and is a very good element.

## 3.5 Tet4

This is a standard 4 node tetrahedral element with three degrees of freedom per node. The **Tet4** element has one integration point. The shape functions are linear. It is not recommended to use only Tet4 elements for the entire mesh because standard, linear tetrahedra are typically much too stiff for structural applications. The **Tet4** is provided primarily for those applications where a mesh may be partially filled with these elements. If a model is constructed of all tetrahedral elements (as by an automatic mesh generator), the **Tet10** is strongly recommended over the **Tet4**.

## 3.6 Tet10

This is a standard 10 node tetrahedral element with three degrees of freedom per node. The **Tet10** uses 4-point integration for the stiffness matrix and 16-point integration for the mass matrix. The shape functions are quadratic. This is a very good element for use in most structural analyses.

## 3.7 QuadT

The **QuadT** is a 4-node quadrilateral shell with membrane and bending stiffness. The element properties and element stiffness and mass matrices are developed by internally generated **Tria3** elements. It is not an optimal element, but is adequate for most applications. A more optimal element is currently under development. See the description of the **Tria3** for details on the element.

## 3.8 Quad8T

The **Quad8T** is an 8-node quadrilateral shell with membrane and bending stiffness. The element properties and element stiffness and mass matrices are developed by internally generated **Tria3** elements. It is not an optimal element, but is adequate for most applications. Shape functions are NOT quadratic. It is compatible with the **Tria6** element, as well as with other elements based on the **Tria3**. See the description of the **Tria3** for details on the element.

## 3.9 TriaShell

The **TriaShell** element has 3 nodes with 6 degrees of freedom (DOF) per node. The **TriaShell** is generated by decoupling the membrane DOF and the bending DOF. Allman's Triangular (AT) element[7] models the membrane DOF, while the Discrete Kirchoff Triangle[8] (DKT) models the bending DOF. These two elements are combined into the **TriaShell** element. It currently supports only isotropic materials. The **TriaShell**, like the **Tria3**, has a single required attribute, `thickness`.

## 3.10 Tria3

The **Tria3** is a three dimensional triangular shell with membrane and bending stiffness. There are 6 degrees of freedom per node. In most respects it is very similar to the **TriaShell**. It is the default element for triangular meshes. The **Tria3** was provided by Carlos Felippa of UC Boulder. It currently supports only isotropic materials. It has a single required attribute, `thickness`, which may be specified in either the exodus file or the text input file.

The element stiffness matrix for triangles consists of the sum of two independent contributions from membrane and bending. These contributions may be arbitrarily scaled using the parameters **membrane_factor** and **bending_factor**. Each of these parameters default to 1.0. They must be specified in the text input file in the block definition.

| Attribute | Keyword | Description |
|---|---|---|
| 1 | thickness | Thickness of the shell |
| 2 | offset | offset for the shell |
| N/A | membrane_factor | scale factor for membrane |
| N/A | bending_factor | scale factor for bending |

The thickness may either be entered in the **Exodus** file, or in the input file. If an attribute is entered in both locations, the value in the input file will be honored. An example element block is shown below.

```
Block 3
        Tria3
        Thickness 0.01
        material 71
        membrane_factor=0  // option to turn off membrane stiffness
End
```

## 3.11 Offset Shells

Any shell may be offset by specifying an offset. This single number is multiplied by the element normal to arrive at an offset vector. The resulting mass and stiffness properties are equivalent to the stiffness generated by translating the shell by the offset vector, and constraining the resulting offset nodes to the untranslated nodes using rigid links. The performance is vastly better than the constraint approach. Note that for curved surfaces there may be modeling issues with offset elements since there is no change in curvature with the change in radius. In the `.inp` file the element offset is specified as,

```
offset=-3.14e-2
```

Offsets may also be specified in the exodus file. For shell elements these are specified in the attributes 2. Note however, that at this time there are few tools to support model building.

## 3.12 HexShell - under development

The 8 noded hexshell is a hybrid solid/shell element. It is meshed as a standard hex element, but the formulation of the element is similar to that of a shell. Unlike a shell element, the thickness is determined by the mesh. But, the element is designed to operate with many of the same features as shell elements even when it becomes very thin. Details of the element formulation are available in a separate report (Ref. 9).

The hexshell has a preferential thickness direction which is essential to it's correct operation. The thickness direction may be specified in any one of three ways.

1. Using the **tcoord**, it may be specified by a coordinate frame.
2. An exodus side set may be attached to one face of all the elements in a block using the keyword **sideset**.

3. Salinas may attempt to determine the thickness direction from the topology. This is the default option (because it is the easiest for the user), but it is also the least robust.

When the element thickness must be determined by the topology, the mesh must follow these requirements. The elements in the block must form a sheet. More than one disconnected portion of the sheet is possible, but all portions must adhere to these requirements.

- Every element in the sheet must have at least two neighbors, e.g. the sheet can't be a single element. NOTE... at this time, this is true for the parallel decomposed mesh as well. The portions of the sheets found in each subdomain can not be a single element. We must be able to eliminate the thickness direction of each element by it's neighbor connectivity.
- The elements in the sheet may vary in thickness, but the sheet must be exactly one element thick.
- The elements must be connected as a single sheet. Thus, if the sheet turns a corner, it must do so gently. The algorithm will fail if any element in the sheet is connected on the top or bottom to another element in the sheet.

The **HexShell** requires a material specification. Optional parameters include the sideset or the coordinate frame and coordinate direction used to determine the thickness direction. The sideset keyword must be associated with a defined sideset in the model. The **tcoord** keyword requires two integer arguments. The first is the ID of the coordinate system referenced. The second is the direction (1,2 or 3) associated with the coordinate system.

| # | Keyword | Arguments | Description |
|---|---|---|---|
| 1 | sideset | ID | sideset to specify thickness direction |
| 2 | tcoord | ID and direction | coordinate frame and coordinate direction |

An example specification follows.

```
Block 88
   HexShell
   sideset 88
   layer 1
     material 1
     coordinate 1
     thickness .4
   layer 2
```

```
      material 2
      coordinate 2
      thickness 0.6
End

BLOCK 89
   HEXSHELL
   tcoord 5 1  // use coordinate frame 5, "x" direction
   material 89
END
```

The formulation of the HexShell supports multiple layers of orthotropic materials. Each layer has an associated material, normalized thickness and coordinate. The coordinate is provided to permit specification of the material coordinate. The thickness specifies the relative thickness of each layer. The total thickness is determined from the element topology, but relative thicknesses for each layer must be specified. If only one layer is specified, then the layer keyword is not required, and the relative thickness is irrelevant (and not required).

## 3.13 Beam2

This is the definition for a Beam element based on Cook's (Ref. 2) development. This beam is similar to the standard Nastran CBAR element. It has no shear contribution. The **Beam2** has 7 required parameters, and an optional offset vector.

| # | old order # | Keyword | Description |
|---|---|---|---|
| 1 | 1 | Area | Area of beam |
| 2 | 5 | I1 | First bending moment |
| 3 | 6 | I2 | Second bending moment |
| 4 | 7 | J | Torsion moment |
| 5,6,7 | 2,3,4 | Orientation | orientation vector |
| 8,9,10 | 8,9,10 | offset | beam offset vector |

No stress or strain output is available for beams. Beams are restricted to isotropic materials. Attributes may either be entered in the **Exodus** file, or in the input file. Attributes in the exodus file must be in the order specified in the table above. If an attribute is entered in both locations, the value in the input file will be honored. Two attribute orderings are currently supported in Salinas because of inconsistencies in preprocessing tools. See the discussion on "OldBeam" in section 2.3.

The following section illustrates the definition of a **Beam2** block.

```
Block 3
   Beam2
   Area 0.71
   I1 .05
   I2 5e-2
   J 0.994
   orientation 1.0 -1.0 0.9
   material 7
 End
```

Beams may be offset by specifying an offset vector. The resulting mass and stiffness properties are equivalent to a the stiffness generated by translating the beam by the offset direction, and constraining the resulting offset nodes back to the untranslated nodes using rigid links. Note that for curved surfaces there may be modeling issues with offset elements, since there is no change in curvature with the change in radius. In the `.inp` file the offset is specified as,

```
offset=-3.14e-2 0.11 0.99
```

Offsets may also be specified in the exodus file. For beams these are specified in the attributes `8, 9` and `10`. Note however, that at this time there are few tools to support model building.

## 3.14 OBeam

These beams are provided by Carlos Felippa of UC Boulder. They are similar to the simple beams of **Beam2**. They use identical parameters. Because of this duplication, these beams will probably be eliminated in the future.

## 3.15 Truss

This is the definition for a **Truss** element based on Cook (Ref. 2). Trusses have stiffness in extension only. The **Truss** has 1 parameter.

| # | Keyword | Description |
|---|---|---|
| 1 | Area | Area of truss |

No stress or strain output is available for trusses.

## 3.16 ConMass

Concentrated masses are used to apply a known amount of mass at a point location. Because many meshing tools build beams as a building block for **ConMass**, the geometry definition may be either a line or a point, i.e. the **Exodus** file element types are **BEAM**, **BAR**, **TRUSS** or **SPHERE**. If a beam is used, all the mass is associated with the *first* node of the beam.

Parameters for the **ConMass** are listed below. Because of difficulties in translation or generation of the model, the parameters found in the exodus file are not normally used for a **ConMass**. This avoids the confusion generated when mass constant defaults may have been taken from beams for example. As a result, all parameters must be specified in the input or the analysis will fail.

This behavior can be tedious however, if many concentrated masses are found in the model, and if the analyst is confident that the attributes are appropriate for these elements. In this case, use the **ConMassA** element. It is identical to the ConMass, but allows attributes.

| # | keyword | Description |
|---|---|---|
| 1 | Mass | concentrated mass |
| 2 | Ixx | $xx$ moment of inertia |
| 3 | Iyy | $yy$ moment of inertia |
| 4 | Izz | $zz$ moment of inertia |
| 5 | Ixy | $xy$ moment of inertia |
| 6 | Ixz | $xz$ moment of inertia |
| 7 | Iyz | $yz$ moment of inertia |
| 8,9,10 | offset | offset from node to CG |

As an example element block,

```
Block 5
        ConMass
        Mass 1000.0
        Ixx 1.0
        Iyy 2.0
        IZZ 1.5
        offset 30.0 40.0 50.0
End
```

Note: While offsets are provided for concentrated masses, their applicability depends on the model. In particular, an offset is meaningless if applied on a node for which there is no rotational degree of freedom. Conceptually, we are attaching the mass on the end of a long stiff beam. If that beam is attached only to a solid, it is free to rotate which is a model error. Salinas eliminates the offset in this case so the model is usable.

## 3.17 Spring

The **Spring** element provides a simple spring connection between two nodes in a model. Note that the direction of application of the spring should be parallel to a vector connecting the nodes of the spring. It is usually preferable to have the nodes of the spring be coincident. Springs are defined in the exodus database using `BEAM` or `BAR` elements.

The **Spring** element has three required parameters (the translational spring stiffnesses). Rotational parameters are supported using the **RSpring** element described in section 3.18. Currently there is no way to attach off-diagonal elements, i.e. there is no $K_{xy}$ spring element. If that is required, a combination of a spring and a multipoint constraint must be used.

Springs can be defined in user defined coordinate systems.

| # | Keyword | Description |
|---|---|---|
| 1 | Kx | translational spring constant in $X$ |
| 2 | Ky | translational spring constant in $Y$ |
| 3 | Kz | translational spring constant in $Z$ |

As an example element block,

```
Block 51
      Spring
      Coordinate 7
      Kx 1e6
      Ky 1.11E7
      Kz 1000
End
```

### 3.17.1 Spring Parameter Values

It is strongly recommended that all three values of the spring constants be nonzero. This is especially important in parallel analysis performed using domain decomposition. Many domain decomposition tools may partition the model such that zero spring constants lead to singular domain stiffness matrices. This is true even if other elements may eliminate the singularity. This can cause the solver (particularly FETI) to fail.

While setting nonzero spring stiffness helps to avoid solver problems, the underlying domain decomposition problems still exist for parallel calculations. At the time of this writing, all available domain decomposition tools have difficulty with linear elements and particularly with springs. This invariably leads to load balance problems, and may introduce other problems. In many cases in large models, it may be better to replace the spring elements by solid element meshes which more accurately represent the physical connection. While there are more degrees of freedom in the calculation, the accuracy is enhanced, and domain decomposition problems are largely eliminated.

## 3.18 RSpring

The **RSpring** element provides a simple rotational spring connection between two nodes in a model. It is usually preferable to have the nodes of the spring be coincident. RSprings are defined in the exodus database using `BEAM` or `BAR` elements.

The **RSpring** element has three required parameters (the rotational spring stiffnesses). It is strongly recommended that all three components have some stiffness. This is particularly important when doing parallel analysis (see the discussion in section 3.17.1). Translational stiffness require the use of the **Spring** element described in section 3.17. Currently there is no way to attach off diagonal elements, i.e. there is no $K_{xy}$ spring element. If that is required, a combination of an **RSpring** and a multipoint constraint must be used.

RSprings can be defined in user defined coordinate systems. The relevant parameters are listed in the table.

| # | Keyword | Description |
|---|---|---|
| 1 | Krx | rotational spring constant in $X$ |
| 2 | Kry | rotational spring constant in $Y$ |
| 3 | Krz | rotational spring constant in $Z$ |

As an example element block,

```
Block 52
      RSpring
      Coordinate 7
      Krx=1e6
      Kry = 1.11E7
      Krz 0.1
```

```
    End
```

## 3.19 Spring3 - nonlinear cubic spring

The **Spring3** element provides a nonlinear spring connection between nodes in a model. Note that the direction of application of the spring should be parallel to a vector connecting the nodes of the spring. It is usually preferable to have the nodes of the spring be coincident. Springs are defined in the exodus database using `BEAM` or `BAR` elements.

The **Spring3** element has nine required parameters (the translational spring stiffnesses). There is no way to attach off diagonal elements, i.e. there are no $K_{xy}$ spring elements. If that is required, a combination of a spring and a multipoint constraint must be used.

The force applied by the **Spring3** is defined as a cubic polynomial in each of the coordinate directions. Thus,

$$F_x = Kx1 \cdot u_x + Kx2 \cdot u_x^2 + Kx3 \cdot u_x^3 \tag{18}$$

For linear analyses, only the first term is used.

Cubic springs may be defined in user defined coordinate system.

| # | Keyword | Description |
|---|---|---|
| 1 | Kx1 | translational linear spring constant in $X$ |
| 2 | Ky1 | translational linear spring constant in $Y$ |
| 3 | Kz1 | translational linear spring constant in $Z$ |
| 4 | Kx2 | translational quadratic spring constant in $X$ |
| 5 | Ky2 | translational quadratic spring constant in $Y$ |
| 6 | Kz2 | translational quadratic spring constant in $Z$ |
| 7 | Kx3 | translational cubic spring constant in $X$ |
| 8 | Ky3 | translational cubic spring constant in $Y$ |
| 9 | Kz3 | translational cubic spring constant in $Z$ |

As an example element block,

```
Block 51
      Spring3
      Coordinate 7
      Kx1 1e6
      Ky1 1.11E7
      Kz1 0
      Kx2 0
      Ky2 0
      Kz2 0
      Kx3 1e4
      Ky3 1.11E5
      Kz3 0
End
```

## 3.20 Dashpot

A dashpot represents a damping term proportional to velocity, i.e. Coulomb friction. Dashpot elements combine a coulomb friction damper with a simple linear spring. The spring is included to avoid singular stiffness matrices when dashpots are connected without springs. Dashpots are currently only used in transient dynamic analyses. For other analyses only the spring term will be used.

The damping factor is the damping matrix entry. It has units of $force{\cdot}time/length$. For a single degree of freedom system with a mass=$M$, the following equation is satisfied.

$$K \cdot u + c \cdot \dot{u} + M \cdot \ddot{u} = f(t) \tag{19}$$

Currently dashpots are defined in the basic coordinate system only. Because they are single degree of freedom elements, the direction must also be defined (i.e. cid=1, 2 or 3). There are three parameters. All are required.

| # | Keyword | Description |
|---|---|---|
| 1 | K | translational linear spring constant |
| 2 | c | damping factor |
| 3 | cid | coordinate direction (1, 2 or 3) |

As an example element block,

```
      Block 51
            Dashpot
            cid=1  // dashpot is in the X direction
            K=1e6
            c=1e5
      End
```

Dashpots may be represented in the exodus file with any linear element. The Truss element most closely mimics the dashpot's single degree of freedom behavior, and may be the best definition for domain decomposition tools.

Caution should be exercised when using dashpots (or any single degree of freedom element). The remaining degrees of freedom must be properly account for, or the system matrices will be singular. There may also be important domain decomposition issues with dashpots. See section 3.17 for a discussion.

## 3.21 Hys

The **Hys** element provides a simple, one dimensional approximation of a joint going through microslip. Many simple joints can be represented by their *hysteresis* loop, a curve in the displacement vs. force plane. The relevant parameters of this element are indicated in the table, and illustrated in Figure 4.

| # | Keyword | Description |
|---|---|---|
| 1 | Kmax | maximum slope of $f$ vs $u$ curve |
| 2 | Kmin | minimum slope of $f$ vs $u$ curve |
| 3 | fmax | maximum possible force |
| 4 | dmax | maximum possible displacement |

The **fmax**, **dmax** pair define the limits of applicability of the element. The element will fail if the internal force exceeds **fmax** or the displacement exceeds **dmax**. The slope of the curve at the origin is **kmax**. It represents the small amplitude response of the system. The slope at the extremum, i.e. at (**dmax,kmax**) is **kmin**.

A **Hys** element uses a Beam or truss element in the exodus file. At the current time, the element may only be defined in the $X$ direction. An example of the salinas input is shown below.

```
BLOCK 2
     Hys
     Kmax 4.5e+7
     Kmin 3.0e6
     fmax 5.92
     dmax 0.9833e-6
END
```

## 3.22 Shys

A **Shys** is the whole joint model developed by Smallwood and is an element which uses a Beam or truss element in the exodus file. The element is a 2.5 dimensional element with an **Shys** element in both the $X$ and $Y$ directions and a linear spring

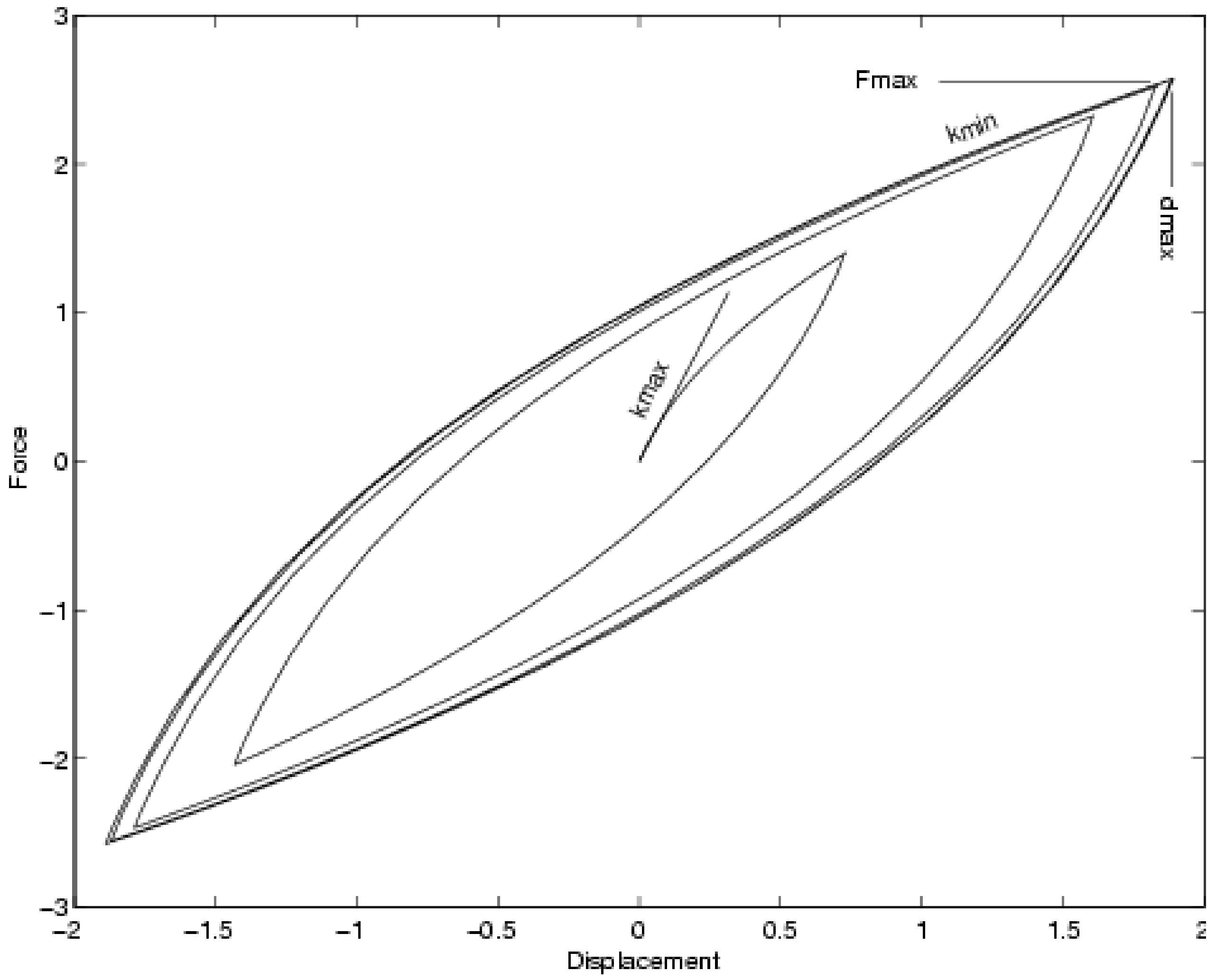

Figure 4: **Hys** element parameters

element in the $Z$ direction. The **Shys** element is assumed identical in both the $X$ and $Y$ directions in this formulation. A coordinate system can be defined to orient the element correctly. An example of the Salinas input is shown below.

| # | Keyword | Description |
|---|---|---|
| 1 | n | Exponent describing slope of force-dissipation curve at very small amplitudes |
| 2 | k | Linear stiffness of Smallwoods element |
| 3 | kNL | Coefficient for non-linear stiffness |
| 4 | kz | Linear translational stiffness in the $Z$ direction |
| 5 | k_r | Linear rotational stiffness (optional, default = 0) |

The **Shys** element does not use the attributes defined in the exodus file for default values of the optional parameters. A detailed discussion of the theory of the **Shys** element as well as how to determine the parameters can be found in the reports by Smallwood (Ref. 10).

```
BLOCK 2
     shys
     coordinate 2
     n = 1.39
     k = 1.3167e6
     kNL = 1.8499e6
     k_z = 1.6e6
     k_rot = 1.e9
END
```

## 3.23 Iwan

A **Iwan** element uses a Beam or truss element in the exodus file. The element is a 2.5 dimensional element with an Iwan element in both the $X$ and $Y$ directions and a spring element in the $Z$ direction. The Iwan element is assumed identical in both the $X$ and $Y$ directions in this formulation. A coordinate system can be defined to orient the element correctly. An example of the Salinas input is shown below.

| # | Keyword | Description |
|---|---|---|
| 1 | chi | Exponent describing slope of force-dissipation curve at very small amplitudes |
| 2 | R | Constant coefficient in distribution |
| 3 | phi_max | Maximum break free psuedo-force |
| 4 | S | Strength of singularity in break free force distribution |
| 5 | alpha | Geometric factor specifying nonuniform spacing of dphi (optional, default = 1.2) |
| 6 | N_elem | Number of slider elements (optional, default = 50) |
| 7 | k_z | Linear translational stiffness in the $Z$ direction |
| 8 | k_r | Linear rotational stiffness (optional, default = 0) |

The Iwan element does not use the attributes defined in the exodus file for default values of the optional parameters. A detailed discussion of the theory of the Iwan element as well as how to determine the parameters can be found in the reports by Segalman (Ref. 11).

```
BLOCK 2
     iwan
     coordinate 2
     chi = -0.58
     R = 585779.
     phi_max = 0.00072925
     S = 671264.
     k_z = 1.6e6
     k_rot = 1.e10
END
```

## 3.24 Gap

Gap elements are modeled after the non-adaptive nastran `CGAP/PGAP` elements. They are intended to provide a simple, penalty type element suitable for modeling simple connections. Note that these elements (like all beam-like elements) when embedded in solid meshes can result in difficult domain decompositions, and lead to load imbalance.

The **Gap** element is inherently nonlinear. In linear analysis, the element behaves approximately like a spring with the stiffness determined by `KL` and a transverse stiffness, KT. The parameters of the element are listed in the table below and shown graphically in Figure 5.

| # | Keyword | Description |
|---|---|---|
| 1 | KU | unloaded stiffness |
| 2 | KL | loaded stiffness |
| 3 | KT | transverse stiffness |
| 4 | U0 | initial gap opening |
| 5 | F0 | Preload, i.e. force at U0 |
| 6 | coordinate | Required coordinate frame. |

The unloaded stiffness, KU, represents the stiffness of the element when the gap is open. It must be greater than zero. The loaded stiffness, KL, represents the stiffness when the gap is closed (as shown in the figure). The stiffness is KL when UA - UB is greater than U0.

The initial gap opening and preload define the corner point in the force/deflection curve as shown in Figure 5. Typically these will be zero.

A gap element provides for transverse stiffness and friction. If the gap is open, there is no shear force. When the gap is closed, the transverse stiffness is KT. In the future, frictional forces will be added.

The coordinate frame is a required attribute of the gap element. The gap open and closes along the $X$ axis of the frame.

The gap element is a simple penalty type element that somewhat mimics the effect of a physical gap. Choice of the value of KL is very important to success of the element. Good values are somewhat in the range of the neighboring element stiffness. Too large a value can lead to matrix condition problems. Too small a value results in excessive softness and penetration in the gap.

Because the element is nonlinear, it has a significant impact on solutions. The current nonlinear solver performs a partial Newton iteration. This means that the tangent stiffness matrix is not updated between iterations. Thus, if KL and KU are quite different, the solver will be using the wrong slope in the newton loop. Many, many iterations may be required for convergence. You may want to turn on the 'timing' option in the echo section (see 2.5) which will put convergence information into the results file.

An example is shown below.

```
BLOCK 2
     GAP
     KL 4.5e+7
     KU 3.0e6
     KT=1e6
     f0 5.92
     u0=0.9833e-6
```

```
        coordinate 5
    END
```

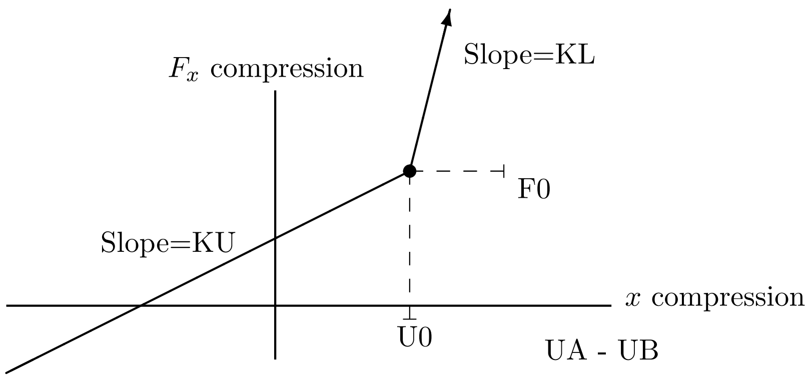

Figure 5: Gap element Force-Deflection Curve

## 3.25 MPC

Multi-Point Constraints (or **MPC**s) are constraint equations applied directly to the stiffness matrix. They are not elements, and are not available from an **Exodus** database. However, in many respects they look like elements, and can be thought of as elements. Some analysis codes treat them as pseudo elements.

All **MPC**s describe constraint equations of the form,

$$\sum_i C_i u_i = 0$$

where $C_i$ is a real coefficient, and $u_i$ represents the displacement of degree of freedom $i$.

Unlike many Finite Element programs, **Salinas** does not support user specification of constraint and residual degrees of freedom (DOF). The partition of constrained and retained degrees of freedom is performed simultaneously by gauss elimination with full pivoting so the constrained degrees of freedom are guaranteed to be independent. Redundant specification of constraint equations is handled by elimination of the redundant equations and issue of a warning. User selection of constrained DOF in Nastran has led to significant headaches for analysts who must insure that the constrained DOF are independent and never specified more than once.

Each **MPC** is specified in the input file with a section descriptor. Note that a separate section is required for each equation (or degree of freedom eliminated). An optional coordinate system may be specified on the input, but must be the first entry in the section. The **MPC** will be stored internally in the basic coordinate system (coordinate 0). The input consists of a triplet listing the global ID of the node, a degree of freedom string, and the coefficient of that degree of freedom. The degree of free strings are $x$, $y$, $z$, $Rx$, $Ry$, $Rz$. They are case insensitive.

In the following example, the $x$ and $y$ degrees of freedom in coordinate system 1 are constrained to be equal for node 4.

```
MPC
    coordinate 1
    4 x 1.0
    4 y -1.0
END
```

Note. Constraints are handled in various ways by the linear solvers. In the serial solver, the dependent degrees of freedom are eliminated before the matrices are passed to the solver. In parallel, we use lagrange multipliers to handle the constraints. There is currently no user control of constraint handling methods.

Note also that there are practical differences between rigid elements (described in the following sections) and constraint equations that are nominally identical. For parallel solutions, we are currently using an augmented lagrange type solution method with the rigid links. This means that terms are added to the stiffness matrix in parallel with the constraints. In most cases, this renders the matrices positive definite, and greatly increases robustness and solution performance with no penalty for accuracy. Thus, rigid links (except RBE3s) are recommended whenever possible in parallel solutions.

Finally note that replacing rigid links with very stiff beams can be a bad thing to do. The condition of the resulting matrices can be severely degraded which can lead to significant loss of accuracy.

## 3.26 RROD

An **RROD** is a *pseudo*element which is infinitely stiff in the extension direction. The constraints for an **RROD** may be conveniently stated that the dot product of the translation and the beam axial direction for a **RROD** is zero. There is one constraint equation per **RROD**.

The **RROD** is specified using beams or trusses in the **Exodus** database, with a corresponding **Block** section in the salinas text input file. No material is required and any number of connected or disconnected **RROD**s may be placed in a block. The following is an example of the input file specification for **RROD**s if the **Exodus** database contains beams in block id=99.

```
Block 99
    RROD
END
```

## 3.27 RBar

An **RBAR** is a *pseudo*element which is infinitely stiff in extension, bending and torsion. The constraints for an **RBAR** may be summarized as follows.

1. the rotations at either end of the **RBAR** are identical,
2. there is no extension of the bar, and
3. translations at one end of the bar are consistent with rotations.

The **RBAR** is specified using beams or trusses in the **Exodus** database, with a corresponding Block section in the input file. No material is required and any number of connected or disconnected **RBAR**s may be placed in a block. The following is an example of the input file specification for **RBAR**s if the **Exodus** database contains beams in block id=99.

```
Block 99
    RBAR
END
```

## 3.28 RBE2

**Salinas** has no support for the Nastran **RBE2** element. However, in most cases there is little difference between the **RBE2** element and a collection of **RBAR**s.

## 3.29 RBE3

The **RBE3** pseudo-element's behavior is taken from Nastran's element of the same name. Note however, that the precise mathematical framework of the Nastran **RBE3** element is not specified in the open literature. This element should act like a Nastran **RBE3** for most applications. The element is used to apply distributed forces to many nodes while not stiffening the structure as an **RBE2** or **RBAR** would. The **RBE3** uses the concept of a slave node. Constraints are specified as follows.

1. The translation of the slave node is the sum of translations of all the other nodes in the element.
2. The rotation of the slave node is the weighted average rotation of all the other nodes about it.

Because all the nodes in an **RBE3** are not equivalent, each **RBE3** requires its own block ID. In the **Exodus** file, all links connecting to a single **RBE3** are defined in a single element block. The input file then specifies that this is an **RBE3** element block, as shown in the example below. If the model requires many **RBE3**s, a separate block will need to be specified for each.

Note: care must be taken to insure that only one node of the **RBE3** has multiple connections to its links. Further, all links in the **RBE3** must be connected to the slave node.

The following is an example of the input file specification for an **RBE3** if the **Exodus** database contains beams in block id=99.

```
Block 99
    RBE3
END
```

## 3.30 Dead

A **dead** element has no mass and no stiffness. It may be of any dimensionality, solid, planar, line or point. Interior nodes to a block of **Dead** elements will not be included in the computation of the model. There are no parameters for **Dead** elements.

## 4 Stress/Strain Recovery

Stresses and strains are recovered at the centroids of the finite elements using standard finite element procedures. Stress and strain recovery is not implemented for 1-D elements. The stresses/strains calculated for shell elements are calculated in element space and not global space.

## 5 Acknowledgements

Salinas is a success based on work by many individuals and teams. These include the following.

1. The ASCI program at the DOE which funded its development.
2. Line managers at Sandia Labs who supported this effort. Special recognition is extended to David Martinez who helped establish the effort.
3. Charbel Farhat and the University of Colorado at Boulder. They have provided incredible support in the area of finite elements, and especially in development of FETI.
4. Carlos Felippa of U. Colorado at Boulder. His consultation has been invaluable, and includes the summer of 2001 where he visited at Sandia and developed the HexShell element for us.
5. Esmond Ng who wrote *sparspak* for us. This sparse solver package is responsible for much of the performance in Salinas and in FETI.
6. The *metis* team at the university of Minnesota. *Metis* is an important part of the graph partitioning schemes used by several of our linear solvers. These are copyright 1997 from the University of Minnesota. Documentation is available at `http://www-users.cs.umn.edu/~karypis/metis/metis/index.html`.
7. Padma Raghaven for development of a parallel direct solver that is a part of the FETI solver.
8. The developers of the *SuperLU* package. This is used in a variety of areas, including a sparse direct complex solver. More information can be obtained at, `http://www.nersc.gov/~xiaoye/SuperLU`.

## References

[1] Schoof, L. A. and Yarberry, V. R., "EXODUS II: A Finite Element Data Model," Tech. Rep. SAND92-2137, Sandia National Laboratories, 1994.

[2] Cook, R. D. and D. S. Malkaus, M. E. P., *Concepts and Applications of Finite Element Analysis*, John Wiley & Sons, third edn., 1989.

[3] Farhat, C. and Roux, F.-X., "A Method of Finite Element Tearing and Interconnecting and Its Parallel Solution Algorithm," *International Journal for Numerical Methods in Engineering*, **vol. 32**, 1991, pp. 1205–1227.

[4] Knupp, P. M., "Achieving Finite Element Mesh Quality Via Optimization of the Jacobian Matrix Norm and Associated Quantities : Part II - A Framework for Volume Mesh Optimization and the Condition Number of the Jacobian Matrix," Tech. Rep. SAND99-0709J, Sandia National Laboratories, 1998.

[5] Aklonis, J. L. and MacKnight, W. L., *Introduction to Polymer Viscoelasticity*, Wiley, 1983.

[6] Ferry, J. D., *Viscoelastic Properties of Polymers*, Wiley, 1980.

[7] Allman, D. J., "A Compatible Triangular Element Including Vertex Rotations for Plane Elasticity Problems," *Computers and Structures*, **vol. 19**, no. 1-2, 1996, pp. 1–8.

[8] Batoz, J.-L., Bathe, K.-J., and Ho, L.-W., "A Study of Three-Node Triangular Plate Bending Elements," *International Journal for Numerical Methods in Engineering*, **vol. 15**, 1980, pp. 1771–1812.

[9] Felippa, C. A., "The SS8 Solid-Shell Element: Formulation and a Mathematica Implementation," Tech. Rep. CU-CAS-02-03, Univ. Colo. at Boulder, 2002.

[10] Smallwood, D. O., Gregory, D. L., and Coleman, R. G., "A three parameter constitutive model for a joint which exhibits a power law relationship between energy loss and relative displacement," in *Shock and Vibration Symposium*, Destin, FL, 2001.

[11] Segalman, D. J., "An Initial Overview of Iwan Modeling for Mechanical Joints," Tech. Rep. SAND2001-0811, Sandia National Laboratories, 2001.

(this page intentionally blank)

# A Salinas Example Input Files

The following sections give examples of **Salinas** input files. Note, case sensitivity of the keywords is ignored unless in quotes. The exception is the **#include** command, where the filename following the command must not be in quotes, but case sensitivity is preserved.

## A.1 An Eigenanalysis Input File

The following input file will output the first four mode shapes to an **Exodus** output file name *hexplate-out.exo*. A results file, *hexplate.rslt*, will not be created since no results have been selected for output in the **ECHO** section.

```
SOLUTION
         eigen
         nmodes 4
         title 'Obtain First Four Mode Shapes For Hexplate'
END

// The f.e.m. is in hexplate.exo
FILE
         geometry_file         'hexplate.exo'
END

BOUNDARY
         nodeset 77
                  fixed
END

LOADS  // loads are unnecessary for eigenanalysis
END

// Only deformations will be output
OUTPUTS
//         maa
//         kaa
//         faa
         deform
//         stress
//         strain
```

```
END

// No results are output to the text log file, *.rslt
ECHO
//  MATERIALS
//  ELEMENTS
//  JACOBIAN
//  ALL_JACOBIANS
//  TIMING
//  MESH
//  mass
//  INPUT
//  NODES
//  FETI_INPUT
//  DISP
//  STRAIN
//  STRESS
//  MFILE
 none
END


// the following element block is hex.
// exodus tells us it is an 8-node hex.
// The default hex is an underintegraged hex.
BLOCK 44
         material 3
         hex8
END

MATERIAL 3
         name "steel"
         E 30e6
         nu .3
         density 0.288
END
```

## A.2 An Anisotropic Material Input File

The following input file is an example of a hexahedral mesh with anisotropic properties.

```
SOLUTION
         eigen
         title 'Example of anisotropic format'
END

FILE
         geometry_file         'anisogump.exo'
END

boundary
        nodeset 4 y = 0
        nodeset 5 x = 0
        nodeset 6 z = 0
end

loads
         // sum of forces on surface should be equal to area
         // imposed forces are additive
         nodeset 1 force = 0.0  0.083333  0.0
         nodeset 2 force = 0.0 -0.041666  0.0
         nodeset 3 force = 0.0 -0.020833  0.0
end


OUTPUTS
//         maa
//         kaa
//         faa
           deform
//         stress
//         strain
END

ECHO
```

```
// MATERIALS
// ELEMENTS
// JACOBIAN
// ALL_JACOBIANS
// TIMING
// MESH
// mass
// INPUT
// NODES
// FETI_INPUT
// DISP
// STRAIN
// STRESS
// MFILE
none
END


// the following element block  is all hex
BLOCK 1
        hex8
        material 1
END

MATERIAL 1
        name "anisotropic gump"
        anisotropic
        Cij
        1.346    0.5769   0.5769  0       0       0
                 1.346    0.5769  0       0       0
                          1.346   0       0       0
                                  0.3846  0       0
                                          0.3846  0
                                                  0.3846
        density 1
END
```

## A.3 A Multi-material Input File

The next example shows the input for an **Exodus** model with many element blocks and materials. Keyword **lumped** in the **SOLUTION** section causes **Salinas** to use a lumped mass matrix instead of a consistent mass matrix.

```
SOLUTION
         eigen
         nmodes 1
         titile 'Multiple block, multiple material example'
         lumped
END

FILE
         geometry_file         'multi.exo'
END

BOUNDARY
    nodeset 1
    fixed
    nodeset 3
    x = 0
    y = 0
    z = 0
    RotY = 0
    RotZ = 0
END

OUTPUTS    // output only displacements to exodus file
     deform
END

ECHO
    none
END

// element block specifications. One such definition per element
// block in the exodus (genesis) database.
BLOCK 1
```

```
         material 2
         Beam2
END

BLOCK 101
         integration full
         wedge6
         MATERIAL 1
END

BLOCK 2
         material 2
END

BLOCK 102
         integration full
         wedge6
         MATERIAL 2
END

BLOCK 3
         material 3
END

BLOCK 103
         integration full
         wedge6
         MATERIAL 3
END

BLOCK 4
         material 4
END

BLOCK 104
         integration full
         wedge6
         MATERIAL 4
END
```

```
BLOCK 5
         material 5
END

BLOCK 105
         wedge6
         integration full
         MATERIAL 5
END

BLOCK 6
         material 6
END

BLOCK 106
         wedge6
         integration full
         MATERIAL 6
END

// material specifications. Extra materials are acceptable, but
// every material referenced in a necessary "Block" definition,
// must be included here.
MATERIAL 1
         name "Phenolic"
         E 10.5E5
         nu .3
         density 129.5e-6
END

Material 2
         name 'Aluminum'
         E 10.0E6
         nu 0.33
         density 253.82e-6
END


Material 3
         name 'foam'
         E 100.
         nu 0.3
         density 18.13e-6
END

Material 4
         name 'HE'
         E 5E5
         nu 0.45
         density 129.5e-6
END

material 5
         name 'Uranium'
         E 30e6
         nu 0.3
         density 1768.97e-6
end

material 6
         name 'wood'
         E 200.e3
         nu .3
         density 77.7e-6
end
```

## A.4 A Modaltransient Input File

The next example shows the input for a **modaltransient** analysis. Accelerations are output to an **Exodus** file *bar-out.exo*. This example has damping, polynomial and linear functions. Also, sensitivities are calculated.

```
SOLUTION
  modaltransient
    nmodes 10
    time_step .000005
    nsteps 100
    nskip 1
    title 'Test modal transient on prismatic bar'
END

FILE
  geometry_file 'bar.exo'
END

ECHO
//  acceleration
END

OUTPUTS
  acceleration
END

BOUNDARY
  nodeset 1
    fixed
END

DAMPING
  gamma 0.001
END

BLOCK 1
  material 1
END
```

```
MATERIAL 1
  name "aluminum"
  E 10e6
  nu .33
  density 2.59e-4
END

LOADS
  nodeset 3
    force = 1. 1. 1.
    function = 3
END

FUNCTION 1
  type LINEAR
  name "test_func1"
  data 0.0 0.0
  data 0.0150 0.0
  data 0.0152 1.0
  data 0.030 0.0
END

FUNCTION 3
  type LINEAR
  name "white noise"
  data 0.0 1.0
  data 0.0001 1.0
  data 0.0001 0.0
  data 1.0 0.0
END

SENSITIVITY
  vectors all
END
```

## A.5 A Modalfrf Input File

The next example shows the input for a **modalfrf** analysis. Accelerations are output to an **Exodus** file *bar-out.frf*.

```
SOLUTION
  modalfrf
    nmodes 10
    title 'Test modalfrf on prismatic bar'
END

FILE
  geometry_file 'bar.exo'
END

frequency
   freq_min 0
   freq_step=10
   freq_max=3000
   nodeset 3
   disp
END

ECHO
//  acceleration
END

OUTPUTS
  acceleration
END

BOUNDARY
  nodeset 1
    fixed
END

DAMPING
  gamma 0.001
END
```

```
BLOCK 1
  material 1
END

MATERIAL 1
  name "aluminum"
  E 10e6
  nu .33
  density 2.59e-4
END

LOADS
  nodeset 3
    force = 1. 1. 1.
    function = 3
END

FUNCTION 2
// this is a smooth pulse with time duration .05
//  it peaks at approximately t=.02 sec with a
//  value of 0.945
  type POLYNOMIAL
  name "poly_fun"
  data 0. 0.
  data 2.0 -8.0e2
  data 0.5 8.9443
END

FUNCTION 3
  type LINEAR
  name "white noise"
  data 0.0 1.0
  data 10000. 1.0
END

SENSITIVITY
  vectors all
END
```

## A.6 A Directfrf Input File

The next example shows the input for a **directfrf** analysis. Displacements are output to an **Exodus** file *bar-out.frf*.

```
SOLUTION
directfrf
END

Frequency
  freq_min = 1000.0
  freq_step = 7000
  freq_max = 5.0e4
  disp
  block 1
End

FILE
  geometry_file 'bar.exo'
END

OUTPUTS
disp
END

ECHO
//
none
END

BOUNDARY
  nodeset 1
    fixed
END

BLOCK 1
  material 1
END
```

```
MATERIAL 1
  name "aluminum"
  G 0.8E+9
  K 4.8E+9
  density 2.59e-4
END

LOADS
  sideset 1
  pressure = -1.0
END
```

## A.7 A Statics Input File

The following example is a **statics** analysis which will output stresses to the **Exodus** output file *quadt-out.exo*.

```
SOLUTION
        statics
        title '10x1 beam of quadt'
END

FILE
         geometry_file         'quadt.exo'
END

BOUNDARY
           nodeset 1
            fixed
END

LOADS
           nodeset 2
            force = 1000.0 1000.0 0.0
END

OUTPUTS
      stress
END

ECHO
  none
END

// the following element block is quadt
BLOCK 1
         material 1
         QuadT
END

MATERIAL 1
         name "steel"
         E 30.0e6
         nu 0.25e0
         density 0.7324e-3
END
```

## B Running Salinas on serial UNIX platforms

On serial unix platforms, **Salinas** is run with a single argument, the ASCII input file.

```
salinas example.inp
```

The log file will be written to *example.rslt* if outputs have been specified in the ECHO section. If outputs have been specified in the OUTPUTS section, a new exodus file will be generated. The file name is derived from the `geometry_file` specified in the ASCII input (see section 2.9).

Visualization of the exodus output results can be accomplished using a variety of `seacas` codes. This includes `blot` (for models with supported element types). Commercial software with exodus preferences is also available. These include *MSC/Patran* and *EnSight*. For more information, contact the authors.

(this page intentionally blank)

## C Running Salinas in Parallel

This appendix gives an example of how to perform an analysis on the Intel Teraflop (**janus**) using **Salinas**. This implies that the execution of **Salinas** will be in parallel. There is some overhead to running in parallel versus serial. Assuming a **Salinas** text input file exists and an **Exodus** file exists which contains the finite element model, the following steps are needed.

1. Decide on how many processors, *nproc*, are needed.
2. Create an input file for **nem_slice**. The partition software can be executed on a workstation to create a load balance file. The name of this file is specified in the input file for *nem_slice*, and usually has a *.nem* extension.
3. Create your workspace on **janus** on /scratch/tmp_?? - where ?? is (currently) your choice of 1 thru 10.
4. Move the **Salinas** input file, **Exodus** file, and load balance file to your work space on **janus**.
5. Create an input file for **nem_spread**. Execution of **nem_spread** (on **janus**) with this input will create *nproc* **Exodus** files from the master **Exodus** file and move them to the locations specified in the **nem_spread** input file.
6. Modify the **FILE** section of the **Salinas** input file to agree with the number of RAID disks available and the location of the subdomain **Exodus** files created by **nem_spread**.
7. Modify the **ECHO** section in the **Salinas** input file using the keyword **subdomain** to indicate which processors should produce text results files. Having all processors output text results files is very slow for large models.
8. Use the **yod** command to run **Salinas** in parallel.
9. Create an input file for **nem_join** to join your results back into one **Exodus** output file.

Each step is detailed in the following paragraphs. Additional information on parallel execution can be found at *http://jal.sandia.gov* under the **SEACAS** documentation link.

## C.1 Number of Processors Needed

Running **Salinas** in parallel requires the user to specify how many processors at a minimum are needed in order to “fit” the problem into available memory on **janus**. For most applications running om compute nodes with 128MB per node, a good rule of thumb is to have approximately 2000 elements/processor. If a finite element model has 1,000,000 elements, use 500 processors.

## C.2 Using Nem_slice to load balance the model

An example of a **nem_slice** input file is, e.g. *junk_slice.inp*,

```
Graph Type = elemental
Decomposition Method = multikl,cnctd_dom
Input ExodusII File = junk.exo
Output NemesisI File = junk.nem
#Solver Specifications
Machine Description = mesh=500
Misc Options = face_adj
#Weighting Specifications
```

This input file will create a load balance file, *junk.nem*, for running **Salinas** on 500 processors. Note, the *face_adj* option is useful for 3-d models to prevent mechanisms from appearing in the decomposed subdomains and is highly recommended for optimal performance.

To create the load balance file, *junk.nem*, simply type

```
prompt> nem_slice -a junk_slice.inp
```

The load balancing software, **nem_slice**, is typically executed on a serial machine such as a workstation. More detailed information on **nem_slice** is available at *http://jal.sandia.gov* under the link to the **SEACAS** documentation.

## C.3 Janus Work Space

To run **Salinas** in parallel, work space on **janus** is needed. On the /scratch space on **janus**, there are 10 temp directories. Simply choose one, and make a directory using your username, as follows.

```
janus> cd /scratch/tmp_1
janus> mkdir $USER
```

After the work space on janus is set up, move the **Salinas** input file, **Exodus** file, and load balance file (*junk.nem*) to it.

## C.4 Using Nem_spread

The load balanced **Exodus** database must be "spread" to *nproc* mini-databases. Each processor reads from its own data file. An example **nem_spread** input file is, e.g. *junk_spread.inp*.

```
Input FEM file                    = junk.exo
LB file                           = junk.nem
Debug                             = 4
------------------------------------------------------------
                Parallel I/O section
------------------------------------------------------------
Parallel Disk Info      = number=18
Parallel file location  = root=/pfs_grande/tmp_, subdir=username
```

Here, `username` must be replaced by the name of the user.

The **Exodus** file and the load balance file need to be defined in the **nem_spread** input file. There are 18 RAID disks currently available on **janus**. These are the number of disks available to which input/output can be performed in parallel. The **FILE** section in the **Salinas** input file needs to have the number of raids defined using the keyword **numraid**. Therefore, for janus, **numraid 18**, must appear in the **Salinas** input file. This number must match the parallel disk info line in the **nem_spread** input file.

If running for the first time on janus, proper directories must be established on the RAID disks. Currently, the raids are setup at */pfs_grande/tmp_??* where ?? is a number between 1 and 18 ( 18 raids ). A few `csh` shell commands can make the required directories.

```
janus> foreach i (1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18)
foreach? mkdir /pfs_grande/tmp_$i/$USER
foreach? end
janus>
```

To execute **nem_spread**,

```
janus> /cougar/bin/yod -sz 4 nem_spread junk_spread.inp
```

This execution of **nem_spread** will spread *nproc* **Exodus** files onto the RAID disks specified in the input file for **nem_spread**. This location must also be specified in the **FILE** section of the **Salinas** input file as follows, assuming your load balance file is *junk.nem* created for 500 processors,

```
FILE
   geometry_file '/pfs_grande/tmp_%d/username/junk.par.500.%.3d'
   numraid 18
END
```

The "`%d`" after *tmp_* is used in **Salinas** in conjunction with the number of RAIDs available. The "`%.3d`" at the end of the line for the **geometry_file** is used in conjunction with how many processors the load balance file was created with. The following table shows what must be used after *junk.par.*`nproc` for various processors requested.

| Condition | Use this |
|---|---|
| $nproc<10$ | "%.1d" |
| $10 \leq nproc<100$ | "%.2d" |
| $100 \leq nproc<1000$ | "%.3d" |
| $1000 \leq nproc<10000$ | "%.4d" |

Since **nem_spread** is a parallel code, **yod** must be used to execute it, using the -sz option to specify how many processors are needed. This number need not agree with the number of processors for execution of the analysis. Typically no more than 20 processors would be used to spread files. The **showmesh** utility can be used to indicate the number of interactive processors available.

## C.5 Salinas FILE Section

If a load balance file *junk.nem* is created for execution of **Salinas** for 500 processors, and the number of raids is 18, then the **FILE** section of the **Salinas** input file must look like the following.

```
FILE
  numraid 18
  geometry_file '/pfs_grande/tmp_%d/username/junk.par.500.%.3d'
END
```

## C.6 Running Salinas

Once the necessary setup has been done, and a parallel **Salinas** code exists in your work space, then

```
janus> cd /scratch/tmp_1/$USER
janus> yod -sz 500 salinas junk.inp
```

This will run **Salinas** in parallel on 500 processors using the input file *junk.inp*.

In practice, only a small number of processors are available interactively on **janus**. To use a larger number of processors, the NQS queuing system must be used. Help is available under the `man` pages on **janus** under the topics `qsub` and `qstat`. To submit an NQS submission, create a small shell script, such as the following.

```
janus> cat run_it
#!/bin/sh
date
cd /scratch/tmp_1/$USER
yod -sz 500 salinas junk.inp
date
```

The NQS job is submitted using `qsub` with a command such as the following.

```
/usr/bin/qsub -lT 90:00 -lP 500 -q snl.day -me run_it
```

This command submits a 90 minute run using 500 processors to the queue `snl.day`. A message will be mailed to you when the run has completed, and output from standard out and standard error will be found in files in your working directory. Status of your run can be obtained using `qstat`. Status of all NQS submissions is available with `qstat -a` or `qstat -av`. Contact *janus-help@sandia.gov* for information on queueing policies and options.

## C.7 Using Nem_join

Once the analysis run has been completed, the output exodus files will need to be recombined into a single file for visualization and processing. **Nem_join** accomplishes this process. A **Nem_join** input file is very similar to the **nem_spread** input file. An example input file is, e.g. *junk_join.inp*.

```
Input FEM file                  = junk.exo
Scalar Results FEM file         = junk-out.exo
Use Scalar Mesh File            = yes
Parallel Results file base name = junk-out.par
Number of processors            = 500
Debug                           = 4
------------------------------------------------------------
                Parallel I/O section
------------------------------------------------------------
Parallel Disk Info      = number=18
Parallel file location  = root=/pfs_grande/tmp_,subdir=username
```

To run **nem_join**, simply do the following:

```
janus> yod -sz 4 nem_join junk_join.inp
```

This will create a file *junk-out.exo* in your current directory by combining all the **Exodus** output files located on the RAID disks. This is a standard exodus file which may be visualized and processed using serial tools.

# D Trouble Shooting FETI Issues

## D.1 Introduction

The Finite Element Tearing and Interconnecting (FETI) solver achieves unprecedented speed and scalability on massively parallel computers. However, it is significantly more complex than a standard direct solver. We discuss a number of the options associated with the solver in the following sections. These options are required to achieve three sometimes-competing goals.

1. Insuring that there is sufficient memory to run on the MP platform.
2. Obtaining the current solution through correct rigid body (or zero energy) identification on the subdomain and on the coarse grid.
3. Tuning the solver to maximize performance.

## D.2 Standard FETI Block

```
FETI
    rbm                    geometric
    preconditioner         dirichlet
    corner_algorithm       1
    corner_dimensionality  6
    corner_augmentation    none
    max_iter               200
    orthog                 1000
    solver_tol             1e-6
    grbm_tol               1e-6
    coarse_solver          sparse
    local_solver           sparse
    precondition_solver    sparse
    prt_summary            yes
    prt_rbm                yes
    prt_debug              2
END
```

## D.3 Memory

The FETI options that directly affect memory usage are listed in the following table. Memory is directly related to the “size” of a subdomain. The number of elements associated with a subdomain can approximately measure the “size”. The topology or connectivity of those elements also directly affects the memory since this determines the local sparse matrix structure.

Large memory allocations occur in the following order with the relative importance listed in parentheses. These operations are only done once for linear static/dynamic and eigen analysis in Salinas.

| | | |
|---|---|---|
| 1. | Preconditioner | (3) |
| 2. | Local Solver | (2) |
| 3. | Coarse Grid | (1) |
| 4. | Orthog vectors | (4) |

### D.3.1 Preconditioner

The lumped preconditioner requires less memory but generally does more iterations than the dirichlet preconditioner which requires more memory. The `precondition_solver` option only affects the memory if the Dirichlet preconditioner is selected. Then the comments in the Local Solver section also apply.

### D.3.2 Local Solver

The skyline solver typically takes more memory than the sparse solver. For small problems (less than 1000 equations), the skyline solver may require less memory than the sparse solver. Generally the skyline solver is the more robust option particularly when the solution may be singular (i.e. eigenvalue analysis on a floating structure).

### D.3.3 Coarse Solver

The corner algorithm, corner dimensionality, corner augmentation, and coarse solver options affect the coarse grid memory requirements. The number of equations in the coarse grid can be found in the solution.data file. Reducing the number of equations in the coarse grid reduces the memory required by the coarse grid.

If your model has shell elements, then corner dimensionality 6 results in more memory than corner dimensionality 3. If your model does not have shells, then this option will not affect memory. Corner dimensionality 6 is generally required for good performance on shell models.

Corner algorithm memory requirements are model dependent and are directly related to the interface topology of the decomposed global model. Typically, corner algorithm 0 results in the smallest coarse grids. This is also the least robust corner algorithm. Corner algorithm 3 is the most conservative corner algorithm and typically generates larger coarse grids. It is recommended to start with corner algorithm 1. If problems arise, change to corner algorithm 3.

Both the skyline and sparse coarse grid solvers are redundantly stored on every processor. The same comments about the skyline and sparse solvers found in the Local Solver section apply here too. The parallel sparse (psparse) solver distributes the coarse grid memory among Ns coarse solver processors. Very large coarse grids can be used with this option. If there are any problems found with the parallel sparse solver, please contact me at khpiers@sandia.gov.

### D.3.4 Orthogonalization (Ortho) Vectors

The number of ortho vectors directly affects the memory requirements of FETI-DP. Generally, you want to select as many ortho vectors as possible given the memory limitations. Ortho vectors decrease the number of iterations required for successive right hand side vectors (eigen/dynamic analysis).

### D.3.5 Options that Affect Memory

```
FETI
         preconditioner [lumped/dirichlet]
         precondition_solver [skyline/sparse]
         orthog 200
         local_solver [skyline/sparse]
         coarse_solver [skyline/sparse/psparse]
         corner_dimensionality [3/6]
         corner_algorithm [0,1,2,3,4]
         corner_augmentation [none/subdomain/edge]
END
```

## D.4 Local Rigid Body Modes

Local rigid body modes (RBMs) refer to the local subdomain stiffness matrix having singularities found during the LDLT factorization and in general the solution will be corrupted if local RBMs are found. The command “prt_rbm yes” in the FETI block will print the number of local RBMs found for each subdomain in your model. Each subdomain is expected to have zero local RBMs. The following steps can be taken if you find a subdomain with a non-zero number of local RBMs.

1. Reduce the tolerance used in the LDLT factorization, For example, the default value for “rbm_tol_mech” is 1.0E-08, then try “rbm_tol_mech 1.0E-12”

2. If this does not remove the local RBMs, then try changing the corner algorithm while holding the previously set tolerance constant. The recommended and

default algorithm is 1. If corner algorithm 1 fails to remove the local RBMs, then try corner algorithm 3.

3. If you have shell elements in the model (and more specifically in the subdomain you have found local RBMs), then "corner dimensionality 6" may be required.

4. If you still have local RBMs, contact me at khpiers@sandia.gov and I'll be happy to look at your specific problem.

## D.5 Global Rigid Body Modes

Global rigid body modes (RBMs) refer to the global stiffness matrix having singularities present. Finding 6 RBMs for a 3D model is expected when performing an eigen analysis with Salinas and the global model does not have any prescribed displacement boundary conditions. FETI-DP can handle this case, but in many cases tolerances have to be adjusted for a particular model.

Finding the incorrect number of RBMs can lead to either stagnation in the FETI solution or the dreaded "relative residual greater than 1" error in Salinas. Troubleshooting this problem can be done in the following fashion.

1. First, determine the expected number of RBMs in your model. Typically in eigen analysis, this is zero (fully constrained), three (2D-floating), or six (3D-floating). The number of RBMs is expected to be zero for transient dynamics.

2. Next, determine how many you are finding with the FETI parameters you have selected. The number of global RBMs are printed to the screen during a Salinas run and printed to the solution.data file. Executing the following UNIX command will find the number of global RBMs found during the last Salinas run. grep "Global RBM" solution.data

3. The parameter "grbm_tol 1.0E-06" will have to be adjusted to find the expected number of RBMs in your model.

4. Decrease grbm_tol if you want to find less global RBMs.

5. Increase grbm_tol if you want to find more global RBMs.

6. For eigen analysis, you may want to use a negative shift (in the Salinas SOLUTION block). Use a shift value equal to the negative of the first anticipated flexible eigenvalue, i.e. $(2\pi f)^2$. This should eliminate all global RBMs, but may slow the solution.

7. If you still have problems with global RBMs, please contact me at khpiers@sandia.gov and I will be happy to help resolve the problem.

# Index